175 PTAS
98
mi computer
CURSO PRACTICO DEL ORDENADOR PERSONAL,
EL MICRO Y EL MINIORDENADOR
NAMAL TYPE & TALK
BRITISH BROADCASTING CORPORATION
MICROCOMPUTER SYSTEM
BREAK
Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen IX-Fascículo 98

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°, 08021 Barcelona
Tel. (93) 209 80 22

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-181-X (tomo 9)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 118512
Impreso en España-Printed in Spain-Diciembre 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

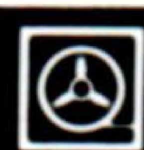
b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

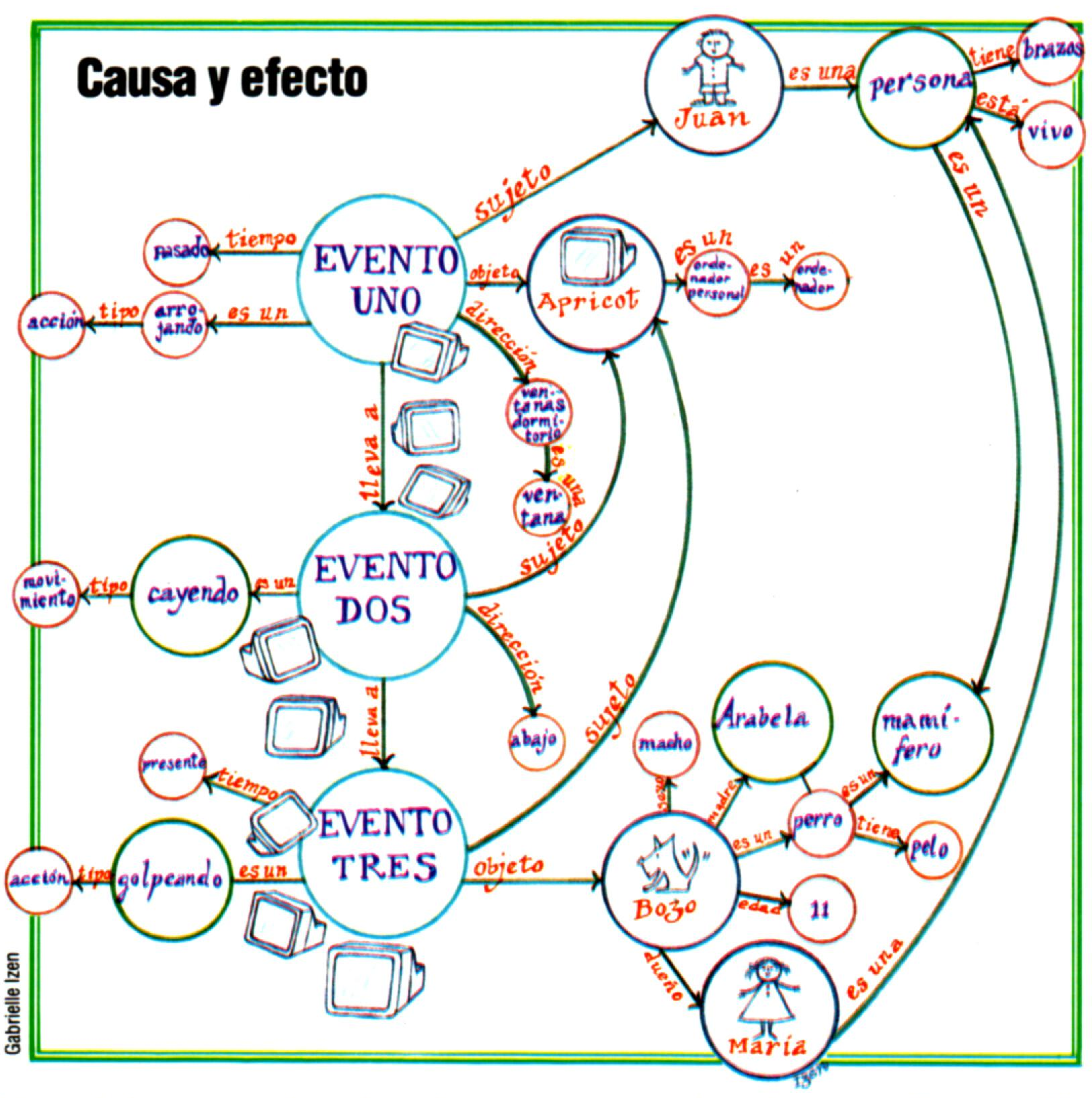
No se efectúan envíos contra reembolso.





El conocimiento

Examinemos algunas de las maneras en que se representa la información en los sistemas de inteligencia artificial



Secuencia semántica
Las redes semánticas se utilizan para conectar acciones y objetos con el fin de describir sus relaciones. La red que vemos aquí representa: «Juan arrojó el ordenador personal por la ventana. Golpeó al perro de María.» Los círculos son los nudos de objetos y las flechas son los arcos de relaciones. A partir de los enlaces semánticos dados se podría deducir, por ejemplo, que «un ordenador golpeó a un mamífero peludo» o que «algo cayó sobre el hijo de Arabela». En un sistema real, la red sería muchísimo más amplia que esta simplificada que ofrecemos aquí

El objetivo de los programadores de inteligencia artificial (AI) es imitar sistemas complejos, como la memoria humana y el raciocinio, tarea en absoluto sencilla. Éste es uno de los motivos por los que hablan de «representación del conocimiento» en vez de «representación de datos».

La mayoría de los programadores poseen una clara idea acerca de lo que se quiere significar con datos. Pero con frecuencia se muestran escépticos a la hora de utilizar la palabra «conocimiento» para describir la información retenida en un ordenador. A la larga, la diferencia entre datos y conocimiento se reduce a una diferencia de énfasis, proveniente de la naturaleza de los problemas con los que se enfrentan los programadores de AI y las soluciones que idean. Es útil analizar los cuatro componentes principales que distinguen al *conocimiento* de los *datos*.

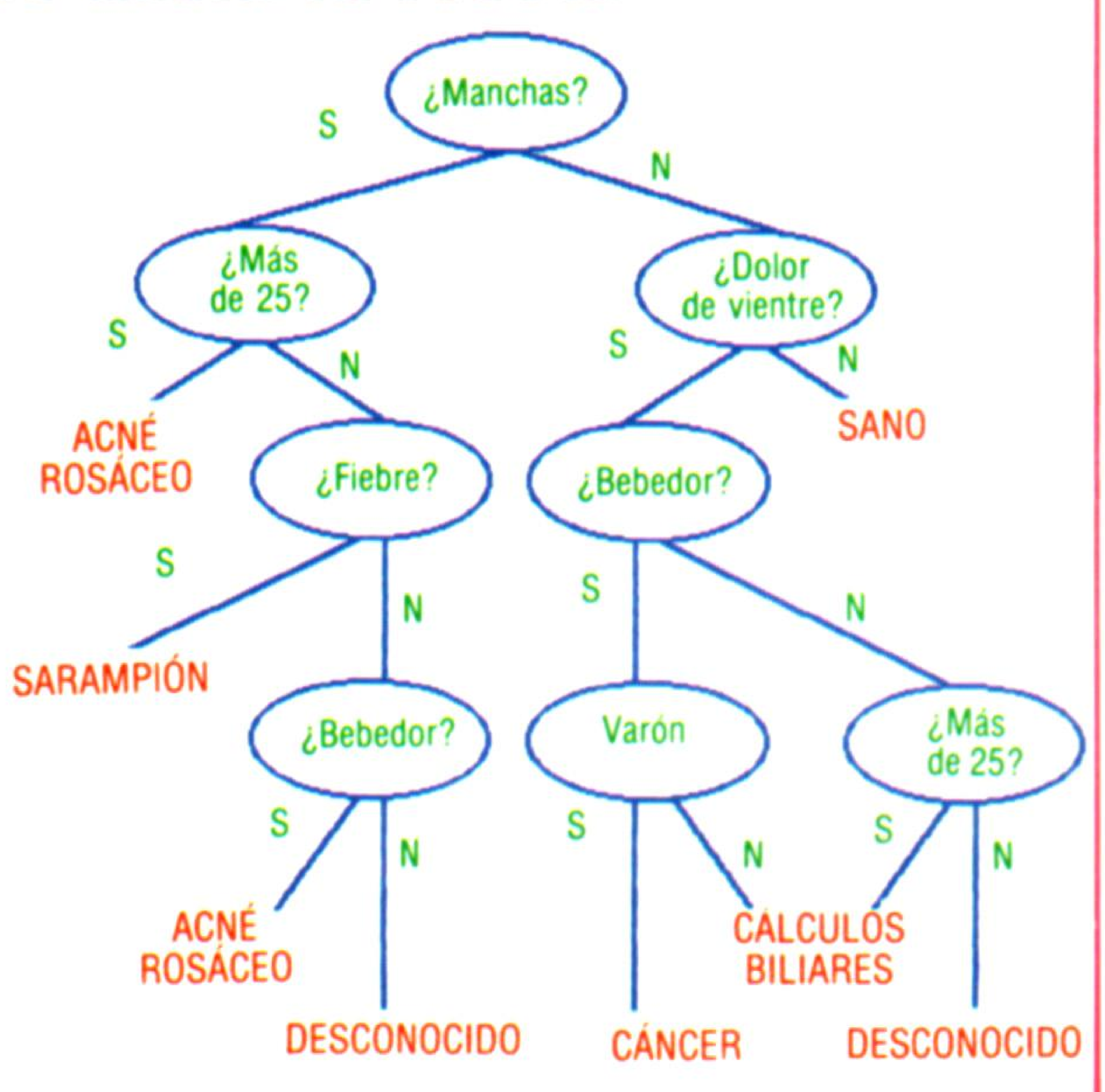
En primer lugar, las representaciones del conocimiento deben ser flexibles en lugar de estáticas. El conocimiento se debe codificar mediante estructuras que pueden reducirse o ampliarse a medida que se ejecuta el programa (utilizando una asignación dinámica de memoria) y no mediante estructuras cuyo tamaño y forma sean fijos para la duración de la ejecución. Lamentablemente, la mayoría de las versiones de BASIC no proporcionan estructuras de datos dinámicas.

En segundo lugar, la representación del conocimiento requiere estructuras de niveles múltiples o estratificadas en lugar de estructuras de un solo nivel. Por ejemplo, las listas pueden contener sublistas; los árboles, contener subárboles; las reglas, aludir a subreglas, y así sucesivamente.

En tercer lugar, las representaciones del conocimiento son típicamente heterogéneas: contienen una mezcla de tipos de datos. Por ejemplo, quizá usted desee establecer un registro para describir a un personaje de un juego de aventuras. El mismo tendría que contener algunas series (un nombre), algunos números (p. ej., edad y altura), señaladores a otros registros (como aquellos de los amigos y

Los caminos hacia la receta

Los árboles de decisión son fáciles de construir y emplear. Sin embargo, se basan en la absoluta precisión de la información utilizada para seguir una cierta ruta a través del árbol. Si los datos son inciertos (como suele ser el caso en la vida real) una representación incorrecta puede conducir a la zona incorrecta del árbol, dando como resultado un diagnóstico erróneo



Caroline Clayton

enemigos de este personaje), reglas que describan el comportamiento de este personaje en situaciones típicas, y muchas otras cosas más. Esto se puede hacer en BASIC, pero no es fácil. Ello se debe a que el principal método para organización de datos (la matriz) sólo puede contener series o números, pero no ambos. Y, ciertamente, no puede contener matrices ni otros objetos de datos exóticos tales como elementos.

Por último, y tal vez lo más importante, las representaciones del conocimiento son activas, mientras que las estructuras de datos son pasivas. Efectivamente, los sistemas basados en el conocimiento, han convertido la separación en dos hileras —programa y datos—, en las tres hileras que son los hechos, las reglas y un motor de inferencias.

Los hechos corresponden claramente a los datos. El motor de inferencias es, obviamente, un programa. Pero las reglas se identifican un poco con ambos. Constituyen *datos activos*. Habiendo esbozado las diferencias clave entre datos y conocimiento, veamos algunas de las formas en que se puede representar y almacenar el conocimiento.

Matrices

La matriz ofrece un método simple para representar el conocimiento. El único problema es que puede resultar demasiado simple para muchas aplicaciones de AI. En el contexto del sistema experto, por ejemplo, imaginemos un paquete simplificado para diagnóstico médico con sólo cuatro diagnósticos (hipótesis) y siete síntomas (evidencias). Sería posible codificar el conocimiento del sistema en forma de una matriz bidimensional (como podemos ver). Este formato de matriz sería adecuado para soportar una estrategia de inferencia sencilla, recogiéndose la evidencia de un elemento por vez en la escala −1 (no), 0 (quizá) y +1 (sí). Una vez producida toda la evidencia, simplemente se la puede multiplicar por los números de las filas correspondientes a cada columna. La columna que registre el total mayor es, entonces, la hipótesis más firmemente sustentada. Surgirían problemas al tratar de ampliar tal representación. Es inflexible: con 200 evidencias y 100 categorías de enfermedades, la matriz habría de contener 20 000 celdas, la mayoría de las cuales estarían vacías. También es demasiado «chata», en cuanto que ignora la estructura jerárquica del conocimiento médico. Las enfermedades se pueden agrupar en tipos. Una vez que un médico sabe que un paciente padece cierta clase de enfermedad, toda una serie de preguntas se vuelven irrelevantes. Una estructura más adecuada sería, en otras palabras, un árbol.

Árboles

Las estructuras arborescentes constituyen una ayuda de incalculable valor para los programadores de AI. Dos aplicaciones importantes para las estructuras arborescentes en el campo de la AI son los *árboles de decisión* y *las jerarquías de herencia*. A modo de ejemplo de la primera estructura arborescente, hemos reorganizado los datos tabulares de nuestro ejemplo anterior en forma de árbol de decisión (véase el diagrama). El problema fundamental de los árboles de decisión en cuanto a la representación del conocimiento es que manipulan muy mal la incertidumbre. Son muy eficientes cuando las pruebas de cada nudo son inequívocas, pero si las respuestas son inciertas, el sistema se puede introducir fácilmente en la parte errónea del árbol. En la vida real muy raramente se puede proporcionar respuestas sí/no definitorias para las preguntas. Los árboles de jerarquía de herencia (véase diagrama de p. contigua) conectan términos y conceptos de modo que la estructura arborescente refleja las relaciones de una forma ordenada. Este tipo de árbol permite realizar inferencias de sentido común cuando se manejan los elementos del árbol.

Redes semánticas

Una red o estructura gráfica es más compleja que un árbol. Un árbol posee un nudo «raíz» especial y todas las ramificaciones se despliegan en una dirección, por lo general hacia abajo. Una red no posee ningún nudo raíz y los enlaces pueden apuntar en cualquier dirección. Las redes semánticas son una clase especial de estructura gráfica que ha sido acogida favorablemente en AI para representar conocimientos acerca del lenguaje y acerca de las acciones y los eventos. En una red semántica, los nudos representan objetos y los arcos o enlaces entre ellos representan relaciones.

La red semántica es una generalización de una construcción del LISP que tuvimos oportunidad de ver en el capítulo anterior: la lista de propiedades. Tanto las listas de propiedades como las redes semánticas codifican la información como pares de valores-atributos unidos a un objeto.

Es básicamente una cuestión de conveniencia; y, de hecho, tanto las listas de propiedades como las redes semánticas se pueden resumir bajo el concepto de *tuples*. Un *tuple-n*, por ejemplo, es simplemente una agrupación de *n* objetos. Las representaciones de objetos-atributos-valores (como redes semánticas y listas de propiedades) emplean ternas, es decir, tuples-3, como en:

Objeto	Atributo	Valor
BOZO	ES-UN	PERRO

Entre la idea de representar el conocimiento como un conjunto de ternas y la noción de «cuadros» hay sólo un corto paso. El cuadro es una estructura muy utilizada en los trabajos de AI, que contiene varias *ranuras*. Cada ranura es como el campo de un registro convencional de un archivo, pero la cantidad de ranuras no es fija.

Por lo tanto, los cuadros son, en esencia, *tuples-n* donde *n* puede variar. La siguiente es una representación tipo cuadro que describe parte de la información de la red semántica que vimos anteriormente, con la adición de algunos hechos más:

```
Nombre:         Evento-3
Tipo:           Golpeando
Tiempo:         Ahora
Sujeto:         PC49
Objeto:         Bozo
Antecedente:    Evento-2

Nombre:         Bozo
Es-un:          —> Perro-prototipo
Dueño:          María
Tipo:           Agente-animado
Sexo:           Macho
Edad:           11
Padres:         (Fido, Arabela)
Implicado-en:   (Evento-3, Evento-7, Evento-20...

Nombre:         Perro-prototipo
Tiene:          Pelo
Sabe:           (Ladrar, Morder, Cazar...)
Patas:          Defecto=4
Ejemplos:       (Bozo, Fido, Arabela, Spot)
Peligroso:      IF pequeño AND dormido OR
                meneando el rabo
                        THEN No
                        ELSE Sí
Hacer:          IF durmiendo THEN dejarlo
```

Podemos ver aquí que las ranuras se pueden llenar con diversos tipos de datos: Edad posee un número, Padres posee una lista, Peligroso tiene una regla de decisión, Es-un tiene un señalador a otro cuadro, y así sucesivamente. La ranura corresponde a los atributos de una lista de propiedades o a los arcos de una red semántica.

Puesto que cualquier estructura suficientemente flexible puede representar la información de cualquier otra clase de estructura, parecería que la elección entre las representaciones es básicamente cuestión de eficacia. No obstante, a algunos investigadores de AI no les agrada el relativismo que esto implica e intentan definir un formalismo que lo abarque todo y que sea más fundamental que los otros. El formalismo que eligen es la lógica de predicado. La lógica como lenguaje de representación es la base del PROLOG, que utiliza un esquema basado en lo que se denomina *cláusulas cuerno*. Por ejemplo:

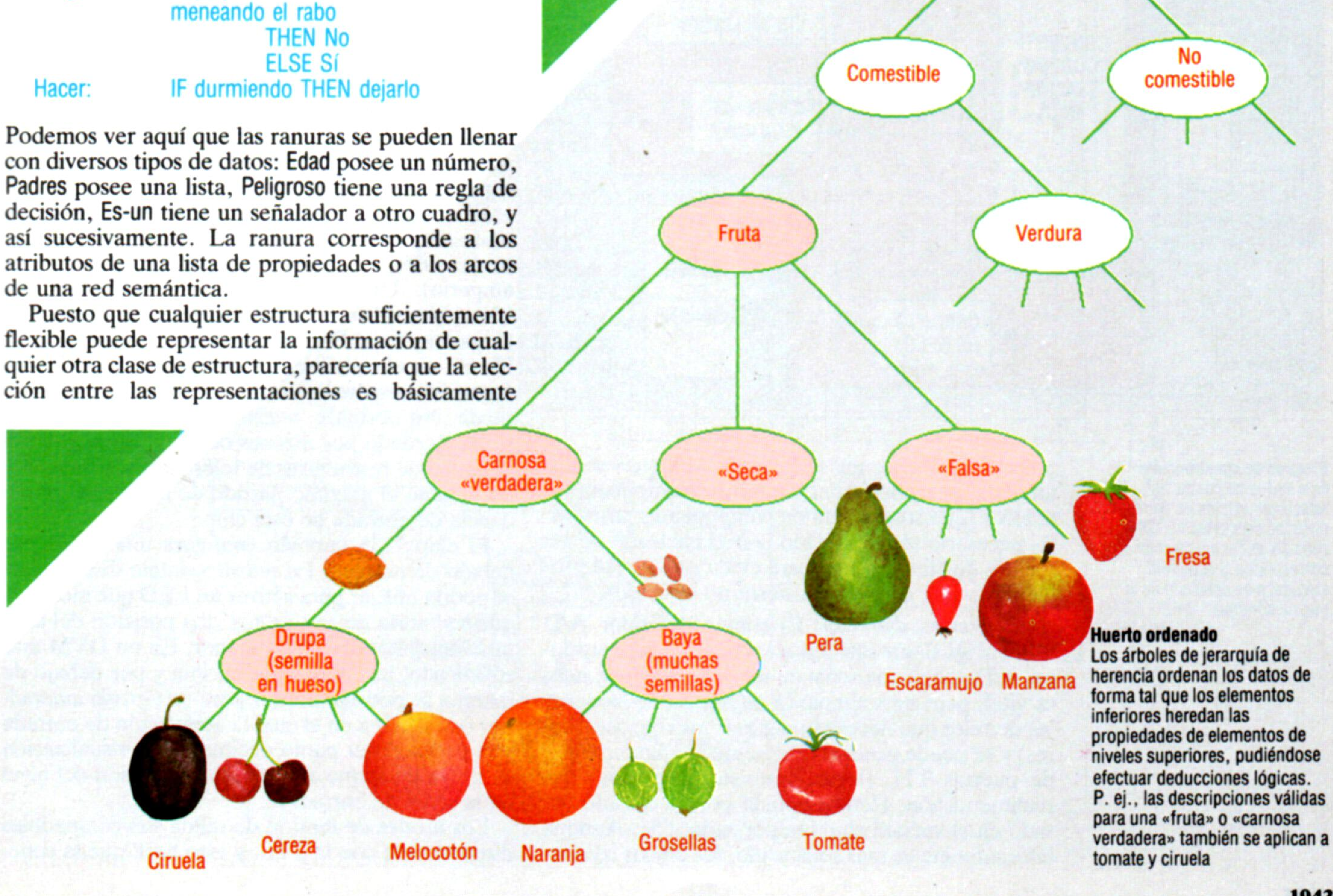
```
peligroso (X):–
        perro (X), no (seguro(X))
seguro (X):–
        pequeño (X), dormido (X)
seguro (X):–
        meneando el rabo (X)
```

es una versión en PROLOG de una de las reglas sobre perros de las estructuras de cuadros anteriores.

En sentido teórico, la lógica es más fundamental que las otras representaciones que hemos analizado. Pero en la práctica, la elección de representaciones viene dictada por consideraciones ajenas a la elegancia teórica.

El ingeniero de conocimiento bien equipado habrá de ser consciente de las variedades de esquemas de representación, que se han utilizado con todo éxito en los proyectos de AI. Adecuar la representación al conocimiento es uno de los artificios de la inteligencia artificial.

Clasificando frutas



Huerto ordenado
Los árboles de jerarquía de herencia ordenan los datos de forma tal que los elementos inferiores heredan las propiedades de elementos de niveles superiores, pudiéndose efectuar deducciones lógicas. P. ej., las descripciones válidas para una «fruta» o «carnosa verdadera» también se aplican a tomate y ciruela

David Davies

De tal palo, tal astilla

En el capítulo anterior estudiamos algunos de los principios de la conversión digital a analógico (D/A) y analógico a digital (A/D). Ahora ofrecemos un detallado análisis del circuito completo del medidor de tensión (DVM) que estamos diseñando en este apartado dedicado al bricolaje

El circuito DVM consta de siete subsecciones o bloques (como se indica en el diagrama). Consideremos cada uno de estos componentes:

- *El atenuador y los interruptores de entrada:* La señal de entrada a medir se alimenta mediante cables con puntas de prueba a los terminales de entrada del DVM, a través del bloque de atenuación y de los conmutadores de entrada. La complejidad del atenuador y de los conmutadores depende de lo amplia que sea la gama de valores a medir (de microvoltios a kilovoltios, p. ej.) y de cuantas unidades eléctricas se midan (voltios CC, voltios CA, corriente CC, ohmios, etc.).
- *La fuente de alimentación:* Este bloque se explica por sí mismo. Todo cuanto se requiere son +5 V y −5 V a una corriente bastante baja, y esto se consigue fácilmente utilizando dos reguladores de tensión de tres terminales, tomando la energía de la red eléctrica o de pilas. Todos los bloques se alimentan, a excepción del bloque atenuador/conmutadores (que sólo consta de componentes pasivos). Es necesario tener cuidado con el cableado de las fuentes de alimentación para evitar que surjan problemas a raíz de bucles a tierra no deseados.
- *El circuito del reloj:* El chip convertidor A/D 7135, al igual que casi todos los tipos de convertidores A/D, exige una señal de reloj. La señal de reloj es vital, pero muy simple (a diferencia de las señales de reloj que necesitan algunos microprocesadores) y se puede generar fácilmente mediante un par de puertas TTL (lógica transistor-transistor) con realimentación. Hemos optado por un circuito basado en el versátil chip temporizador 555. Aunque internamente es más sofisticado, los únicos componentes externos necesarios son dos resistencias y un condensador.
- *La interface del ordenador:* Este bloque es una opción extra. El DVM básico se diseñó como unidad independiente, como alternativa de bajo costo a los caros DVM digitales existentes en el mercado. Sin embargo, si usted desea que su ordenador pueda tomar lecturas directas del entorno (en voltios, ohmios, grados de temperatura o lo que sea), entonces se puede añadir este bloque. No afecta a los otros sistemas de circuitos y es opcional.
- *El convertidor A/D:* Éste es realmente el corazón del DVM. Utiliza un único chip configurado para tener una sensibilidad básica de dos voltios (o, para ser precisos, de 1,9999 voltios). Las tensiones medidas más altas se atenúan mediante el atenuador de entrada, de modo que el chip 7135 no recibirá jamás una tensión de entrada superior a dos voltios. Este chip ofrece numerosas ventajas sobre otros convertidores A/D de un solo chip. Convierte la tensión de entrada en una salida BCD (*binary coded decimal:* decimal codificada en binario) que al ordenador le resulta más fácil de leer que las salidas codificadas en siete segmentos de los chips A/D diseñados para activar directamente visualizadores LCD o LED de siete segmentos. Aparte de esto, el chip 7135 posee otras varias configuraciones que lo hacen especialmente atractivo para un proyecto de esta naturaleza. Es sumamente preciso; la precisión básica es de ±1 puntos sobre 20 000 y puede dar lecturas para cuatro lugares decimales en el margen de dos voltios. Posee una lectura de cero garantizada en la visualización para una entrada de 0 V. Tiene *indicación de autopolaridad,* de modo que se pueden leer tensiones tanto positivas como negativas sin tener que invertir las puntas de prueba. Tiene dos formas de indicar una entrada por sobre la escala (unna entrada de más de 1,9999 voltios): ya sea haciendo parpadear la visualización o bien utilizando una línea de señal separada que puede activar un LED para que indique una situación de superación de margen.

El chip posee, asimismo, una impedancia de entrada muy elevada; típicamente la corriente de entrada es de 1 pA (una millonésima, o 1×10^{-12}, de amperio). Una impedancia de entrada tan alta como ésta aplica al circuito que está midiendo una carga muy baja. Por consiguiente, el drenaje de corriente en el instrumento medido no hace caer de forma importante la tensión que se está tratando de medir. No obstante, nuestro atenuador de entrada se ha diseñado por razones de simplicidad y disponibilidad de resistencias de tolerancia restringida, y no obtiene el máximo partido de la altísima impedancia de entrada de este chip.

El chip 7135 también configura una salida por debajo de margen. En nuestro simple diseño, ésta se podría utilizar para activar un LED que mostrara que se habría de seleccionar una posición del conmutador para una escala menor. En un DVM más sofisticado, las salidas por encima y por debajo de margen se podrían utilizar para una *escala automática* (un sistema en el cual la atenuación de entrada y la posición del punto decimal en la visualización se sitúan de forma automática en función del nivel de la señal de entrada).

Los niveles de tensión de salida son compatibles directamente con la TTL, y esto hace que la cone-

Bloques del DVM

Bloques de construcción
Este esquema ilustra las relaciones existentes entre las diversas secciones del DVM. La zona del esquema de color rojo corresponde al esquema circuital más detallado de la página contigua

xión con interface a microordenadores sea comparativamente simple. Aún más importante es el hecho de que se proporcionan varias señales específicamente para simplificar la conexión con interface al ordenador. Éstas incluyen una línea STROBE para sincronizar la transferencia de datos a los enclavamientos externos (registros que pueden retener datos sin necesidad de suministrar una entrada constante), una línea de entrada RUN/HOLD que le permite al ordenador «congelar» la lectura u operar de forma independiente, y una línea BUSY que le dice a la máquina si las otras salidas son válidas o no. Las cuatro líneas de salida BCD sólo poseen valores significativos en ciertos momentos durante el proceso de conversión A/D, y una línea BUSY activa indica que las lecturas de las líneas de salida no son fiables.

Ante todo, el chip 7135 es sumamente versátil. Muchas de las señales de entrada y de salida disponibles se pueden ignorar, pero existen y están listas para usar si se requiere un sistema más sofisticado.

• *El activador del visualizador:* Dado que el convertidor A/D 7135 proporciona una salida BCD, no se puede utilizar directamente para activar el visualizador. Afortunadamete, un chip individual muy económico es todo cuanto se necesita para conectar con interface el convertidor A/D a un visualizador apropiado. El chip 7447A, por el cual hemos optado, puede convertir una señal BCD en la combinación correcta de siete señales necesaria para iluminar correctamente un LED único de siete segmentos. La salida del 7135 es multiplexada. Es decir, cada dígito de la visualización comparte las cuatro señales de salida BCD, pero éstas sólo son válidas para uno de los cinco dígitos por vez. Al mismo tiempo, las señales de dígitos válidos del chip 7135 indican qué digito se está produciendo en BCD en cada momento. Estas señales de dígitos válidos se utilizan directamente para conmutar transistores, habilitando uno de los LED por vez. Dado que las visualizaciones de dígitos se encienden y se apagan con tanta rapidez, las cinco visualizaciones parecen brillar al mismo tiempo.

• *El visualizador:* En la actualidad las LCD están de moda, en especial para los instrumentos portátiles que funcionan a pila. Sin embargo, los LED todavía poseen algunas ventajas, y hemos optado por una visualización de LED. Los LED son algo más simples que las LCD para las conexiones con interface, particularmente cuando se trata de conmutación de puntos decimales. Los LCD requieren una señal de CA de plano posterior, y en el circuito DP (*decimal point:* punto decimal) se requiere un sistema adicional de circuitos lógicos.

Convertidor kernel
La base de nuestro diseño DVM es el chip convertidor A/D 7135. Este chip lleva a cabo sus operaciones en respuesta a señales de reloj externas que le proporciona el chip 555 de la parte inferior del esquema. El 7135 produce un valor digital en forma BCD que se puede utilizar en conjunción con un chip 7447A, multiplexor activador del visualizador, y cinco transistores de conmutación para iluminar un visualizador LED de 4½ dígitos. En próximos capítulos iremos ofreciendo detalles de construcción completos. Aconsejamos a los lectores que no intenten empezar la construcción en esta etapa, puesto que el diseño no está completo todavía

Conversión A/D y visualización

Kevin Jones

Control del reparto

Para ejercer dominio sobre los personajes podemos recurrir a dos métodos distintos

En nuestro primer capítulo de esta serie vimos cómo los personajes controlados por ordenador del juego *Suspect,* de Infocom, podían desplazarse de un escenario a otro, aparentemente «a voluntad». El movimiento, sin embargo, es una de las características menos importantes de una entidad controlada por ordenador, y el programa proporciona ejemplos excelentes de la interacción jugador/personaje más relevante en su sintaxis de instrucciones. Al jugar a *Suspect,* se pueden utilizar instrucciones que permitan dirigirse a los personajes de numerosas formas diferentes. El programa, por ejemplo, aceptaría todas estas entradas:

Michael, ¿dónde está Veronica?
Acusar al coronel Marston de asesinato
Llamar a mi abogado
Johnson, háblame de ti

Es obvio que existe una esfera de acción para que usted se comunique con los otros invitados en la mansión Ashcroft. Pero lo que hace que el juego resulte particularmente atractivo son los numerosos mensajes que se generan en respuesta a los intentos del jugador por mantener conversaciones amables. Aunque los personajes tienden a ser ligeramente repetitivos, aun así dan prueba de una insólita coherencia y sentido en sus respuestas:

Michael, hábleme del caballo
«*Lurking Grue* es el saltador premiado de Veronica. Realmente es un animal muy hermoso.»
Marston, hábleme del caballo
«No sé nada acerca de él que pueda interesarle.»

Además, los personajes no sólo se pueden comunicar con el jugador sino que también pueden hablar entre sí. En realidad, no es probable que éste llegue muy lejos a menos que se pasee por el salón de baile e intervenga en conversaciones tales como:

Michael se une a la conversación. «Recuerdo un semental negro que el año pasado se vendió a un precio muy alto. Probablemente haya superado las cien mil.» El coronel Marston lo mira con indignación y dice: «Tengo buena memoria para los números. El precio máximo del año pasado fueron noventa y dos mil.» Cochrane le lanza una mirada a Michael, sintiéndose traicionado. «Tonterías», dice enfadado…

Resumiendo, las «personas» de *Suspect* satisfacen varias de las exigencias que suelen estipularse para unos auténticos personajes interactivos:

1. Pueden desplazarse de un escenario a otro.
2. El jugador puede dirigirse a ellos y obtener una respuesta con sentido.

También pueden recoger y dejar objetos; de hecho, en cierto momento del juego ¡un personaje llega realmente hasta el punto de esconder un objeto dentro de otro objeto! Por último, los personajes pueden dirigirse al jugador sin ser requeridos en ese sentido. Existen muchos ejemplos de ello durante el juego, produciéndose los más obvios cuando usted es arrestado por el detective (si no ha conseguido hallar al culpable).

Antes de pasar al siguiente paso (programar nuestra propia rutina de manipulación de personajes) es interesante mencionar brevemente un inconveniente del diseño de un juego que depende mucho del comportamiento y las acciones de los personajes controlados por ordenador. La trama de *Suspect,* por ejemplo, exige que en algún momento «alguien» sea asesinado y que, en otros momentos del juego, los personajes lleven a cabo acciones significativas que, en caso de que el jugador las observe o las deduzca, permitan resolver el misterio.

Por este motivo, los personajes no pueden conducirse de forma totalmente aleatoria. Esto significa que, tras ejecutar varias veces el programa, el jugador enseguida caerá en la cuenta de que, por ejemplo, el personaje A estará en el lugar B en el momento C realizando la acción D. Más adelante veremos que esta necesidad de que las personas realicen acciones específicas para poder cumplimentar la trama, requiere una especial consideración al construir una rutina eficiente para manipulación de caracteres. En *El Hobbit,* por ejemplo, los personajes manifiestan en su conducta un nivel considerable de azar. Esto podría significar que están algo limitados en cuanto a su capacidad para comportarse coherente e inteligentemente, pero por regla general un nivel de azar elevado le confiere a un programa una atmósfera propia de «vida real».

En *Suspect,* por otra parte, existe una sensación (después de haberlo jugado varias veces) de *déjà vu* cuando la Reina de las Hadas sale del salón de baile para limpiarse unas gotas de vino que le han derramado sobre el vestido, sólo para acabar como un cadáver en el despacho. El programador encontrará que equilibrar el grado de azar de la conducta de un personaje con la necesidad de coherencia y pertinencia, constituye uno de los aspectos más difíciles de la programación de una buena aventura.

Dos consideraciones

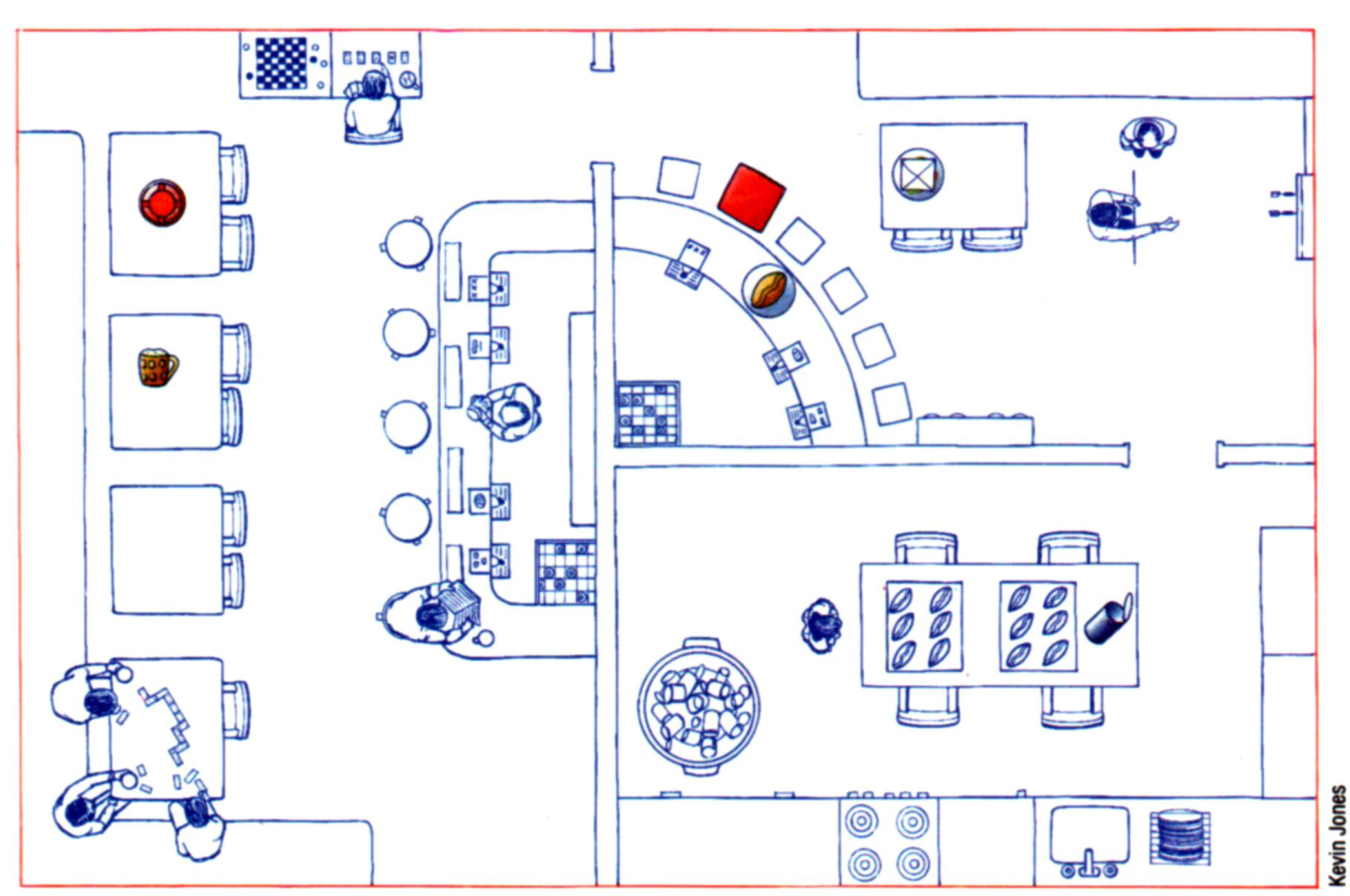
Habiendo analizado *Suspect* de forma bastante detallada, consideremos nuestras propias rutinas. La primera decisión a tomar es de carácter general: ¿cuán complejo queremos que sea nuestro programa? Al diseñar una aventura, por lo general resulta bastante fácil decidir, por ejemplo, cuántos escenarios queremos que haya; pero, cuando se trata de los personajes, las combinaciones son casi infinitas. Un juego como *Suspect* exige grandes cantidades de almacenamiento de datos sólo para los mensajes que utilizan los personajes, por no hablar de las rutinas para controlarlos. Infocom aborda este problema escribiendo sistemas basados sólo en disco, cargando los datos cuando es necesario. Pero para la mayoría de los usuarios europeos los discos constituyen un lujo y, por tanto, todos los programas deben ejecutarse enteramente en RAM.

INICIALIZAR variables, visual., etc.
VISUALIZAR descr. escenarios, etc.
INPUT instrucción
LLAMAR manipul. personajes
DECODIFICAR entrada
EJECUTAR instrucción
BUCLE PRINCIPAL

Caroline Clayton

Posición estratégica
Este diagrama de flujo de un típico juego de aventuras muestra una posible posición para nuestra rutina manipuladora de personajes. Incorporada en el programa en este punto del flujo de control, la rutina se ejecutará cada vez que el jugador pulse ENTER tras haber digitado una instrucción





Operadores locales
Nuestra rutina manipuladora de personajes se desarrollará en Dog and Bucket, un *pub* inglés de dudosa reputación, famoso por su cerveza de tonel y sus excelentes empanadas caseras. Aquí vemos la disposición de los escenarios utilizados en la rutina, junto con las posiciones iniciales de algunos de los objetos más importantes. En el próximo capítulo le mostraremos cómo programarlos en la rutina y le añadiremos personajes a nuestra ilustración a medida que se presenten

Éste no es un factor tan limitador como podría parecer, tal como veremos en el próximo capítulo cuando examinemos *Sherlock,* de Melbourne House, un juego basado en RAM con una manipulación de personajes bastante sofisticada. No obstante, puesto que conviene que las cosas sean moderadamente sencillas, diseñaremos el programa para manipular un máximo de 7 personajes.

La segunda pregunta que es necesario responder no es tan directa; pero, nuevamente, es necesario contestarla antes de iniciar la programación. Existen dos métodos para controlar los personajes de un juego *(síncrono* y *asíncrono)* y hemos de decidir cuál de los dos utilizar. Para explicar cabalmente el significado de estos términos y las implicaciones que tendrá nuestra elección, es preciso refrescar la memoria respecto a qué hace exactamente un juego de aventuras y cómo lo hace.

A partir del diagrama se aprecia que cada vez que el jugador digite una entrada y pulse ENTER, el ordenador la decodificará, llamará a la rutina requerida, actualizará las variables del sistema necesarias, imprimirá mensajes, y luego retornará para aguardar otra entrada. La pregunta a la que hemos de responder es: ¿en qué lugar de esta estructura colocaremos a nuestro manipulador de personajes?

Volviendo a los términos que mencionábamos anteriormente, un sistema síncrono de personajes ejecutaría la rutina de manipulación de personajes en una etapa fija del programa (como vemos en el diagrama). Aquí, cada vez que el jugador pulse ENTER se llamará a la subrutina manipuladora de personajes. Éste es un sistema síncrono, y *Suspect* representa un buen ejemplo de este tipo de programas. Ofrece ciertas ventajas para el jugador; por ejemplo, si usted se aparta del teclado para ir a servirse una taza de té, tendrá la total seguridad de no ser atacado (tal vez) por otro personaje controlado por el ordenador mientras permanezca en la cocina. Por la misma razón, significa que el ritmo del juego tiende a depender excesivamente de las entradas del jugador.

Por el contrario, un sistema asíncrono por lo general será activado por interrupciones y pasará el control al manipulador de personajes de vez en cuando, independientemente de si usted se halla o no frente al teclado. Éste es el sistema que utilizan *El Hobbit* y *Valhalla,* en los cuales, si el jugador se reclina en la silla y observa la pantalla, verá personajes yendo y viniendo, llevando su propia vida de forma bastante independiente. Este sistema posee la ventaja de conferirle «atmósfera» al juego, pero también es más difícil de programar. En particular, usted ha de tener cuidado en asegurarse de que no exista ningún conflicto variable entre su manipulador de personajes y el programa principal, ya que de lo contrario podría encontrarse con graves problemas.

Utilizar rutinas activadas por interrupciones desde BASIC es muy difícil, a menos que el jugador posea un Amstrad o un micro MSX: por este motivo adoptaremos para nuestra rutina el sistema síncrono. Sin embargo, diseñaremos el programa de modo tal que usted pueda reclinarse en su asiento y mirar cómo se ejecuta sin necesidad de ir introduciendo instrucciones.

Estamos ahora en condiciones de iniciar la programación. Aunque el propósito es construir un módulo transportable que se pueda adaptar para ejecutarlo con juegos de aventuras, como nuestros programas *Digitaya* y *El bosque encantado,* aun así necesitamos proveer un entorno en el cual probar la rutina manipuladora de personajes. Por lo tanto, crearemos una aventura sencilla con tres escenarios en los cuales puedan moverse nuestros personajes, y con numerosos objetos a manipular.

Dado que el programa está diseñado para poner de relieve a nuestros personajes interactivos, desarrollaremos la acción en un jovial lugar de encuentro: el *pub* Dog and Bucket (El perro y el cubo).

Datos básicos (V)

Continuamos con nuestro exhaustivo análisis del mapa de memoria del Commodore 64

ETIQUETA	DIRECCIÓN HEXA	POSICIÓN DECIMAL	DESCRIPCIÓN
SA	00B9	185	Dirección actual secundaria
FA	00BA	186	Número dispositivo actual
FNADR	00BB-00BC	187-188	Puntero: nombre actual del archivo
ROPRTY	00BD	189	RS-232 salida paridad/temp. cassette
FSBLK	00BE	190	Cuenta bloques lectura/escritura cassette
MYCH	00BF	191	Buffer palabras en serie
CAS1	00C0	192	Enlace motor cinta
STAL	00C1-00C2	193-194	Dirección inicio entrada/salida
MEMUSS	00C3-00C4	195-196	Temps. carga cinta
LSTX	00C5	197	Tecla actual pulsada: CHR$(n)0 = No hay tecla
NDX	00C6	198	Número de caracteres en buffer teclado (cola)
RVS	00C7	199	Flag: imprime caracteres invertidos: 1 = sí; 0 = no empleados
INDX	00C8	200	Puntero: fin de línea lógica en INPUT
LXSP	00C9-00CA	201-202	Posición cursor X, Y inicio INPUT
SFDX	00CB	203	Flag: imprime caracteres con *shift*
BLNSW	00CC	204	Activa parpadeo cursor: 0 = cursor en flash
BLNCT	00CD	205	Temporizador: cuenta el cursor *toggle*
GDBLN	00CE	206	Carácter bajo el cursor
BLNON	00CF	207	Flag: último parpadeo cursor *on/off*
CRSW	00D0	208	Flag: INPUT o GET desde teclado
PNT	00D1-00D2	209-210	Puntero: dirección actual línea pantalla



Dos «conversadores»

Démosle una mirada a dos nuevos sintetizadores de voz y escuchemos qué nos dicen

Los sintetizadores de voz para ordenadores han experimentado un constante progreso durante estos últimos años, y en la actualidad algunos de estos avances están empezando a implementarse en micros personales. El tipo de sintetizador más antiguo tenía en su memoria una lista de «palabras». Cuando recibía algún texto a pronunciar, comparaba la palabra que había recibido con las que tenía en la memoria. Tras hallar una pareja, llamaba a una rutina que producía la serie de sonidos que, unidos entre sí, formaban la palabra requerida. El sintetizador de voz moderno es bastante más sofisticado. En estos sistemas, el usuario puede digitar cualquier frase y el ordenador «pronunciará» las palabras.

Los problemas inherentes a la obtención de una síntesis de voz correcta y amplia utilizando el enfoque texto-habla son enormes. La complejidad de lenguaje es de por sí suficiente para desbordar aun el programa más sofisticado. Tomemos, por ejemplo, en inglés, la palabra *read* (leer). Este vocablo se puede pronunciar tanto «red» (leí, leído), como «rid» (leer, lee), según el contexto. Ejemplos como éste significan que los microordenadores pueden, en el mejor de los casos, tomar una palabra e intentar la pronunciación correcta. Sin embargo, aunque de ningún modo son perfectos, los nuevos sistemas pueden proporcionar resultados notables.

El sistema opera leyendo en un tampón una serie de caracteres ASCII. Luego se toma cada palabra y se coteja con una serie de «reglas» gramaticales que ha establecido el programador. Partes de la palabra corresponderán a ortografías particulares para las cuales se ha establecido una regla. Por ejemplo, en inglés, una de éstas podría ser que si hay una vocal seguida de una consonante seguida de «ie» o «y», luego la primera vocal debe producir un sonido largo. Mediante la implementación de esta regla, la palabra *vary* (variar) se pronunciaría correctamente. No obstante, el problema del idioma, y en particular del inglés, es que su implementación no está normalizada. En consecuencia, al encontrarse con una palabra como *many,* la regla del sintetizador de voz determinará que la pronunciación correcta de la palabra es «meini». Por lo tanto, muchos programadores incluyen asimismo una lista de excepciones comunes a las reglas, aunque, por supuesto, es imposible incluirlas todas en el limitado espacio de que se dispone en un microordenador. Aun cuando el ordenador tuviera un espacio ilimitado de almacenamiento, el tiempo de procesamiento necesario ocasionaría demoras inaceptables para la producción de la voz.

Las dos máquinas descritas aquí son Namal Type & Talk, para el BBC Micro, y el Amsoft Speech Synthesiser, para ordenadores Amstrad: ambas utilizan el sistema de «reglas» para decodificar el habla.

Type & Talk

Type & Talk es el último de una larga línea de periféricos para el BBC Micro que parecen estar dirigidos al sector educativo. El dispositivo viene en una caja metálica color ante, similar al del ordenador. Type & Talk se puede conectar en interface con el ordenador mediante puertas Centronics o RS423, ya que en la parte posterior de la máquina se proporcionan facilidades para los dos formatos, junto con un conmutador para pasar de una a otra. En la parte posterior de la máquina también hay instalado un conector jack miniatura, que permite conectar el sintetizador a un sistema de altavoz externo.

La parte frontal del sistema Type & Talk alberga un interruptor *on/off,* una rejilla para el altavoz interno y el control de volumen. En el interior del dispositivo hay dos placas de circuito impreso; una de ellas contiene el transformador de potencia y la lógica del sintetizador de voz, mientras que la placa hija contiene los circuitos amplificadores de sonido y el chip codificador de voz. Observando primero la placa principal, aparte de chip PIA y la lógica correspondiente, el Type & Talk posee asimismo una EPROM de ocho Kbytes que contiene el firmware del diccionario y las reglas gramaticales. El procesamiento que se requiere para decodificar la información proveniente del ordenador, y antes de pasarla al chip sintetizador de voz, lo lleva a cabo un procesador NEC D780C (que es, esencialmente, un chip Z80). Para almacenar la información de entrada antes de que se la procese, el Type & Talk posee dos Kbytes para RAM en la placa que actúan como un tampón. En la placa

NAMAL TYPE & TALK
CHIP DE VOZ
Votrax SC01A
MODALIDADES
De conversión directa texto-voz y de fonemas
GENERACION DE SONIDO
Amplificador y altavoz en la placa
INTERFACES
Interfaces Centronics y RS423. Dispone de un conector jack miniatura para auriculares o para un sistema de altavoz externo
VENTAJAS
Produce la pronunciación exacta de gran número de palabras. Debido a que el software se basa en ROM, esto deja al ordenador libre para otras aplicaciones
DESVENTAJAS
Muestra tendencia a recalentarse, lo que origina una distorsión de la voz. Su precio no contribuye a que sea un sintetizador de voz demasiado asequible



Crispin Thomas

Con vocación pedagógica
El Namal Type & Talk es un sintetizador de voz diseñado para usar con el BBC Micro. El dispositivo tiene su propio procesador y software almacenados en ROM. Claramente diseñado con vistas al mercado educativo, en el cual el BBC Micro es importante competidor, la máquina es de construcción sólida, su software es excelente, y proporciona una introducción a la síntesis de voz a la vez entretenida e informativa

AMSOFT SPEECH SYNTHESISER

CHIP DE VOZ	SP0256
MODALIDADES	De conversión texto-voz y de alófonos
GENERACION DE SONIDO	Altavoces gemelos que, aunque pueden ser estéreo, reproducen en mono
INTERFACES	Bus de ampliación que se instala en la puerta para disco flexible del ordenador, con un cable flotante a la puerta para *hi-fi*. Jacks miniatura gemelos para los altavoces
VENTAJAS	Una auténtica conversión texto-voz a buen precio
DESVENTAJAS	La lista de reglas gramaticales no es tan amplia como la de Type & Talk. Adolece de restricciones de hardware impuestas por el ordenador

madre también hay incorporados ocho conmutadores DIP que controlan la velocidad en baudios a la cual se manipula la información de entrada, y otros controles para el intercambio de señales de control y selección de paridad. Al menos una tercera parte de la placa principal está ocupada por el transformador de potencia, que los fabricantes han optado por incluir en la placa.

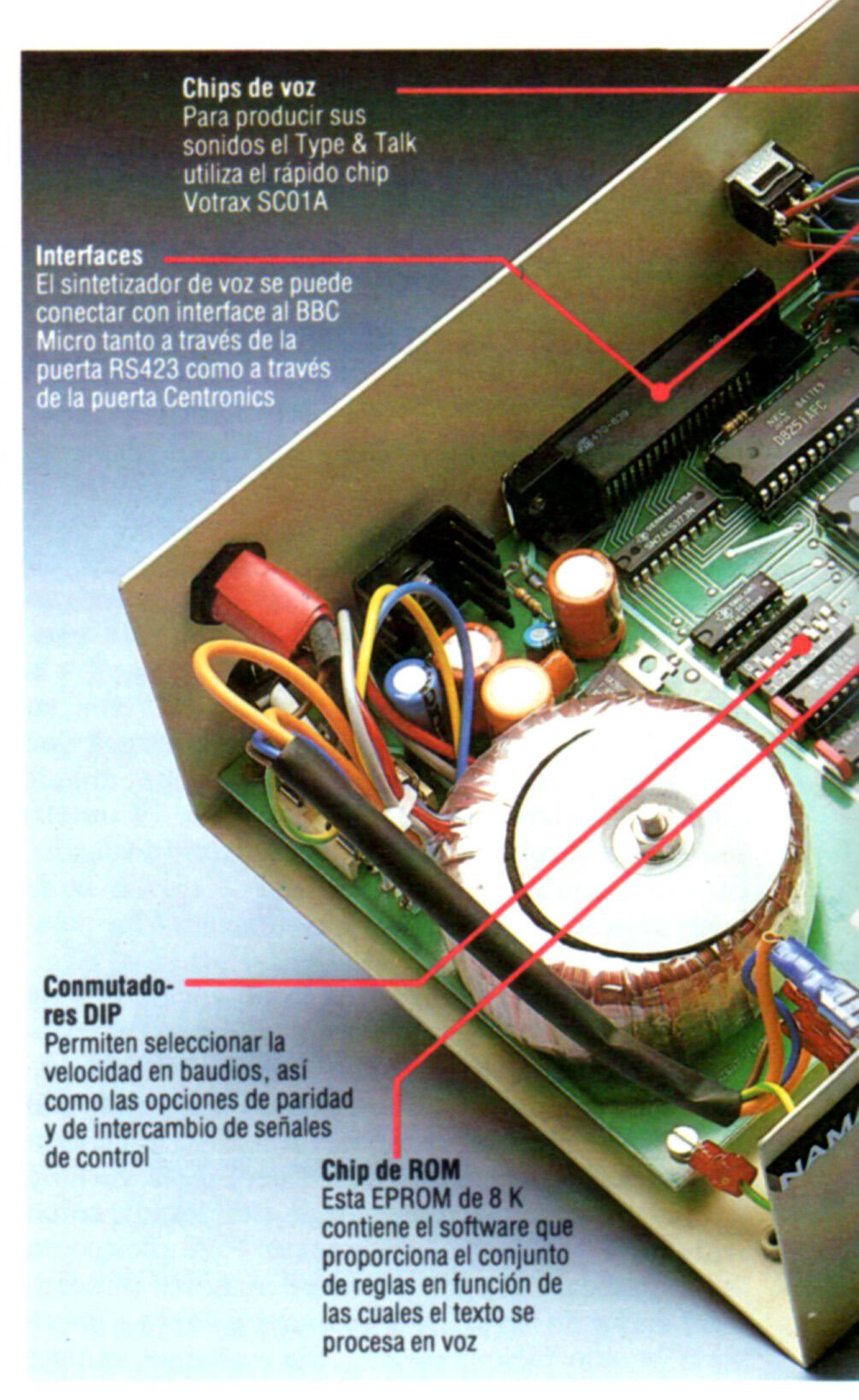
Si bien siempre es preferible tener la fuente de alimentación dentro de una máquina, para reducir la cantidad de cables flotantes, los problemas que entraña tal medida no se han superado por completo en este caso. Después de tener en marcha el Type & Talk durante unas pocas horas, la máquina muestra indicios de sobrecalentamiento, lo que afecta la calidad de la voz producida. La placa hija contiene el amplificador de sonido y un «potenciómetro preajustado» instalado con una tapa atornillada que permite que el usuario regule la velocidad a la cual se produce el habla en el altavoz. La verdadera síntesis de voz la lleva a cabo el chip de voz Votrax SC01A, de gran velocidad, presente en muchos otros sintetizadores de voz para ordenadores.

Una vez conectado el sistema, es sumamente simple producir voz desde el dispositivo. Al encender el Type & Talk, el dispositivo dice Ready Master, indicando que está listo para ser usado. Dado que el ordenador recibe las señales provenientes del sintetizador, en forma de códigos ASCII, enviados a través de los activadores de impresora, el ordenador debe estar configurado para producir un «eco» de impresora de lo que aparece en la pantalla. En el BBC Micro esto se hace digitando VDU 2 o, simplemente, enviándole al Type & Talk un carácter CTRL B. A partir de entonces, todas las frases que se digiten en la pantalla serán pronunciadas por el sintetizador de voz.

Ya hemos visto cómo el sistema Type & Talk decodifica las palabras recibidas desde el ordenador, como también las limitaciones impuestas por las restricciones de hardware. Por consiguiente, para obtener la pronunciación correcta de una palabra a menudo es necesario alterar la ortografía, y para conseguir la articulación correcta a menudo se requiere muchísima inventiva. Por ejemplo, el firmware interpreta la palabra *bough* (rama), que se pronuncia «bau», de la misma forma que la pala-

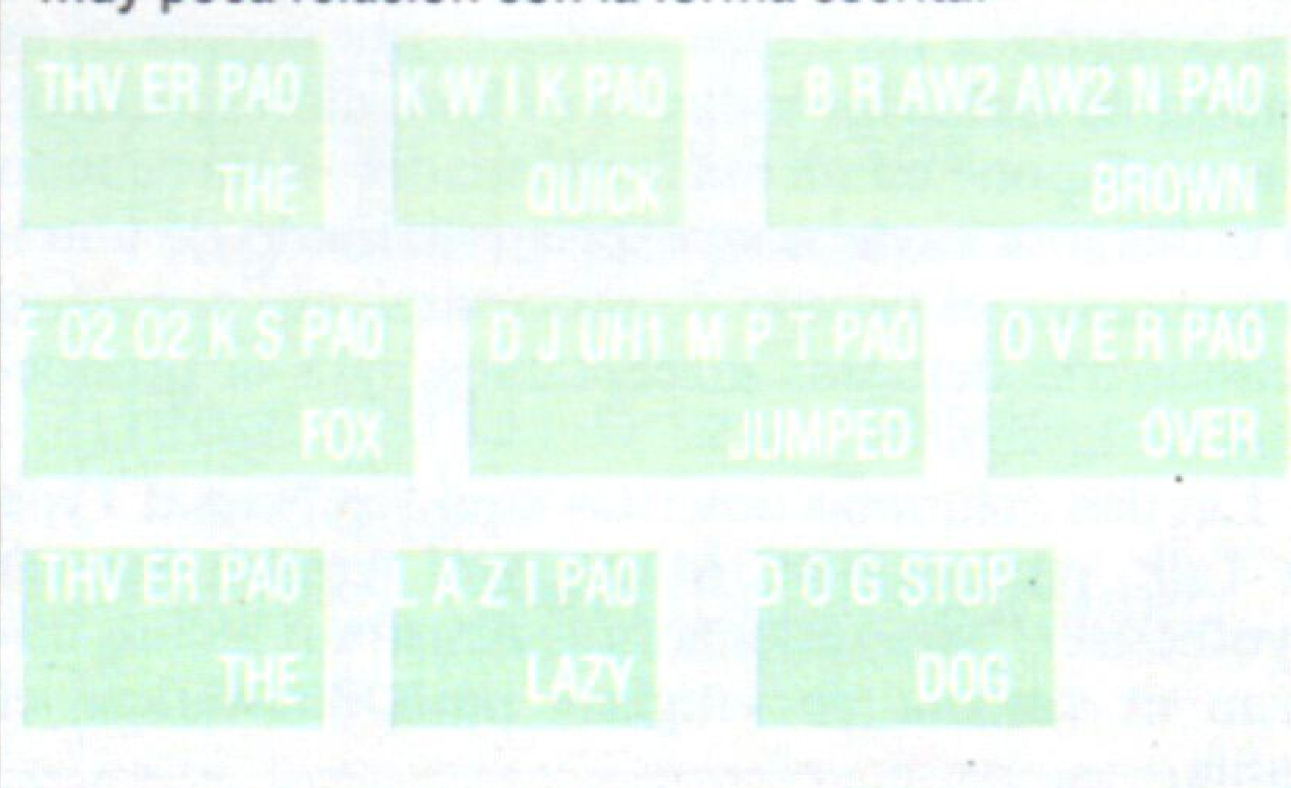


Voz amiga
El precio del Amsoft Speech Synthesiser es mucho más asequible que el del Type & Talk; sin embargo, utiliza la misma técnica de «reglas» que éste. El software se carga (LOAD) desde cassette y el texto se convierte en patrones de habla a través del procesador del propio ordenador. A pesar de no ser tan ambicioso como Type & Talk, el dispositivo de Amsoft está pensado como primera máquina para el usuario de un ordenador personal

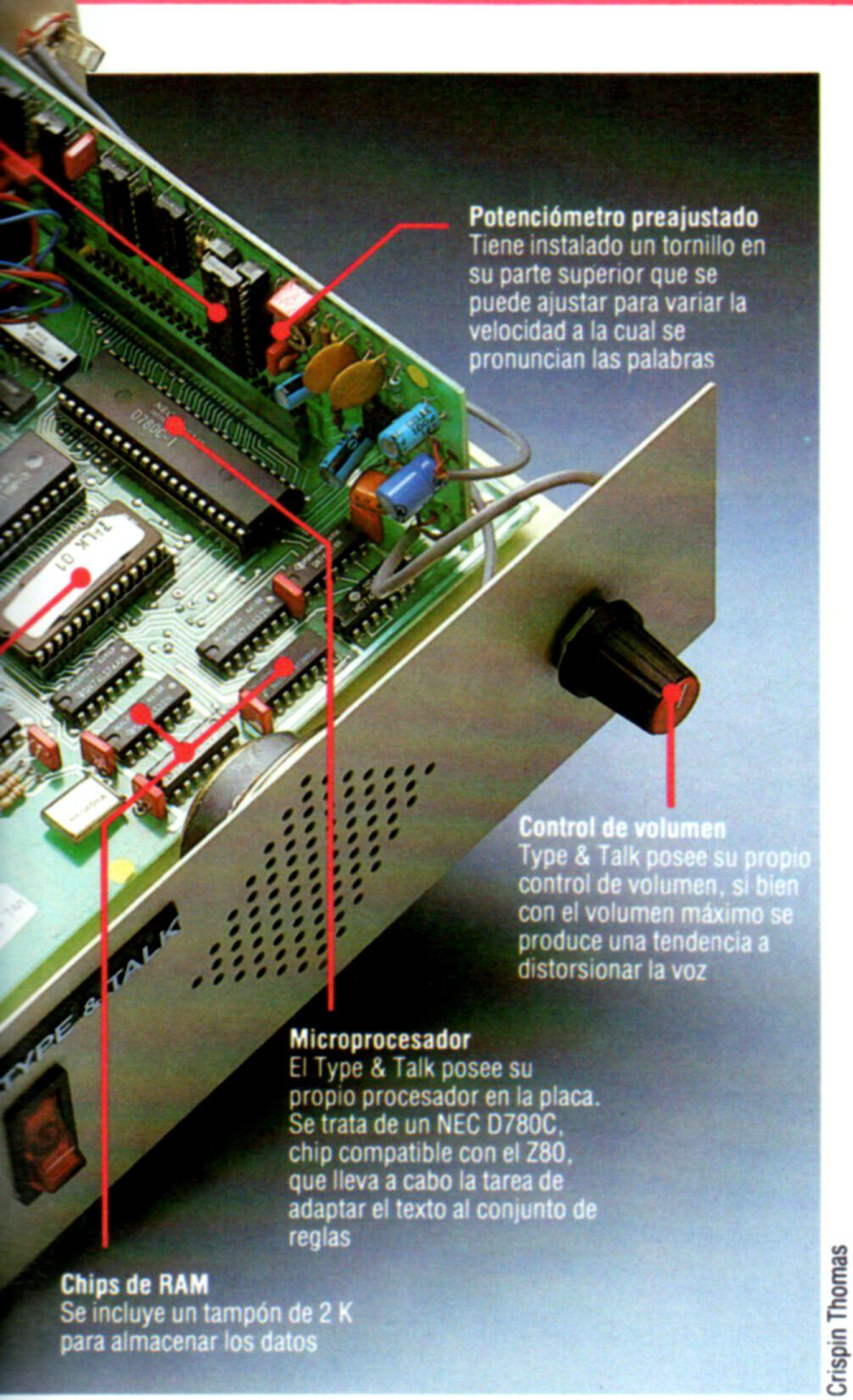


Hablar con facilidad

Al contrario que en la lectura, en la cual podemos reconocer palabras diferentes por la forma en que están escritas, el reconocimiento de las palabras habladas depende de sonidos separados y distintos: los «fonemas». Mientras que el agrupamiento de varios fonemas representa el «grueso» de una palabra, los «alófonos» introducen ligeras diferencias y modificaciones (según el contexto y el lugar en el que se pronuncien las palabras). Por ejemplo, el fonema CH se utiliza tanto en *chip* como en *batch*, pero las pronunciaciones son sutilmente distintas. El alófono, en consecuencia, altera ligeramente el fonema CH para facilitar el reconocimiento de las palabras. Como se puede observar en el siguiente ejemplo, la transcripción fonética a menudo guarda muy poca relación con la forma escrita:







Crispin Thomas

bra *cough* (tos) y, por tanto, la pronuncia «bof». El uso de la ortografía alternativa *bow* (que, según como se pronuncie, puede significar «proa» o «arco») tampoco es de ayuda, puesto que ésta se interpreta para que rime con *low* (bajo). La pronunciación correcta se obtiene escribiendo «bou».

Se puede conseguir un énfasis sobre sílabas determinadas, mediante el envío de caracteres de control entre las palabras a pronunciar. Estos caracteres de control se distinguen de los caracteres ASCII comunes precediéndolos con un !. Un ejemplo de esto lo constituye la palabra HEL!InLO, donde I significa «entonación» y n corresponde a un número entre 0 y 3.

Otro uso de los caracteres de control que le proporciona al usuario muchísimo más control sobre la pronunciación es !P, que coloca al sintetizador de voz en modalidad de fonemas. Los fonemas son una serie de caracteres que corresponden a todos los sonidos que se efectúan en inglés. (Usted puede observar ortografías fonéticas en los diccionarios, donde por lo general se consignan entre paréntesis tras una entrada en negrita.) Enviando una serie de fonemas, en teoría se puede construir cualquier palabra con cualquier acento.

Speech Synthesiser

El Amsoft Speech Synthesiser es una máquina muy diferente. Se trata de una unidad individual que se enchufa en la puerta para disco flexible de los ordenadores Amstrad. La conversión texto-voz la efectúa el propio procesador del Amstrad, suministrándose el diccionario y las reglas en software basado en cassette. Las instrucciones se envían al sintetizador de voz de una forma algo distinta a la del BBC Micro. Hay nueve instrucciones que permiten operar el sintetizador desde BASIC; éstas, al igual que el sistema operativo de disco Amsoft, están implementadas como ampliaciones del sistema residente (RSX). El problema de este sistema es que para que se les puedan pasar parámetros a las RSX, primero hay que convertirlos en series. Por ejemplo, |SAY (con |ECHO constituye las dos instrucciones para conversión texto-voz directa) exige que antes de poder pasar una oración al sintetizador de voz, primero se la ha de implementar como una serie. Afortunadamente, la instrucción |ECHO, que reproduce mensajes en pantalla a través del sintetizador, posee el traspaso de parámetros incorporado.

Comparación de calidad

Ni el sonido ni la calidad de la conversión texto-voz del Amsoft Speech Synthesiser son tan elevados como los del Type & Talk. Esto tal vez no resulte sorprendente, porque el dispositivo de Amsoft es considerablemente más barato que el del periférico para el BBC Micro. El Amsoft Speech Synthesiser posee muchísimas menos reglas, y están implementadas con menor rigor que en el Type & Talk. El resultado es que muchas palabras que en éste se pronuncian correctamente, en el sintetizador de Amsoft se pronuncian mal. Por ejemplo, *able* («eibl») se pronuncia «able» y *cough* («cof») se pronuncia «couf». No obstante, con una ortografía imaginativa se pueden superar estas dificultades.

Otra instrucción importantísima implementada en el Speech Synthesiser es |APHONE. Esta instrucción, al igual que !P en el sistema Type & Talk, permite construir palabras a partir de sus componentes de sonido básicos. Sin embargo, el sistema Amsoft utiliza alófonos como sus «bloques de construcción», en lugar de fonemas (los alófonos son los sonidos que forman parte de los fonemas, aunque en la práctica hay muy poca diferencia). La instrucción |APHONE va seguida por una serie de números, cada uno de los cuales corresponde a un sonido diferente. Este sistema carece de la prontitud del sistema de fonemas del Type & Talk y significa que el usuario debe ir buscando constantemente el número pertinente del sonido que se requiere, pero resulta bastante fácil acostumbrarse a ello.

En la actualidad, muchos de los problemas que entraña la síntesis de voz ya se han superado en gran medida, a pesar del hecho de que los sintetizadores de voz continúan indefectiblemente sonando como a través de embudos. Pero aún persisten dificultades para idear un programa que tenga en cuenta todas las variaciones que se producen en el habla. Es evidente que la tecnología actual no está a la altura de la tarea, si bien la naciente CD-ROM (*compact disk ROM:* ROM en disco compacto) podría representar un seguro paso hacia la solución de los problemas del almacenamiento de «reglas». Sin embargo, estos dos paquetes son representativos del estado actual de la técnica y constituyen un buen punto de partida para la tecnología en que se basarán los futuros sistemas.

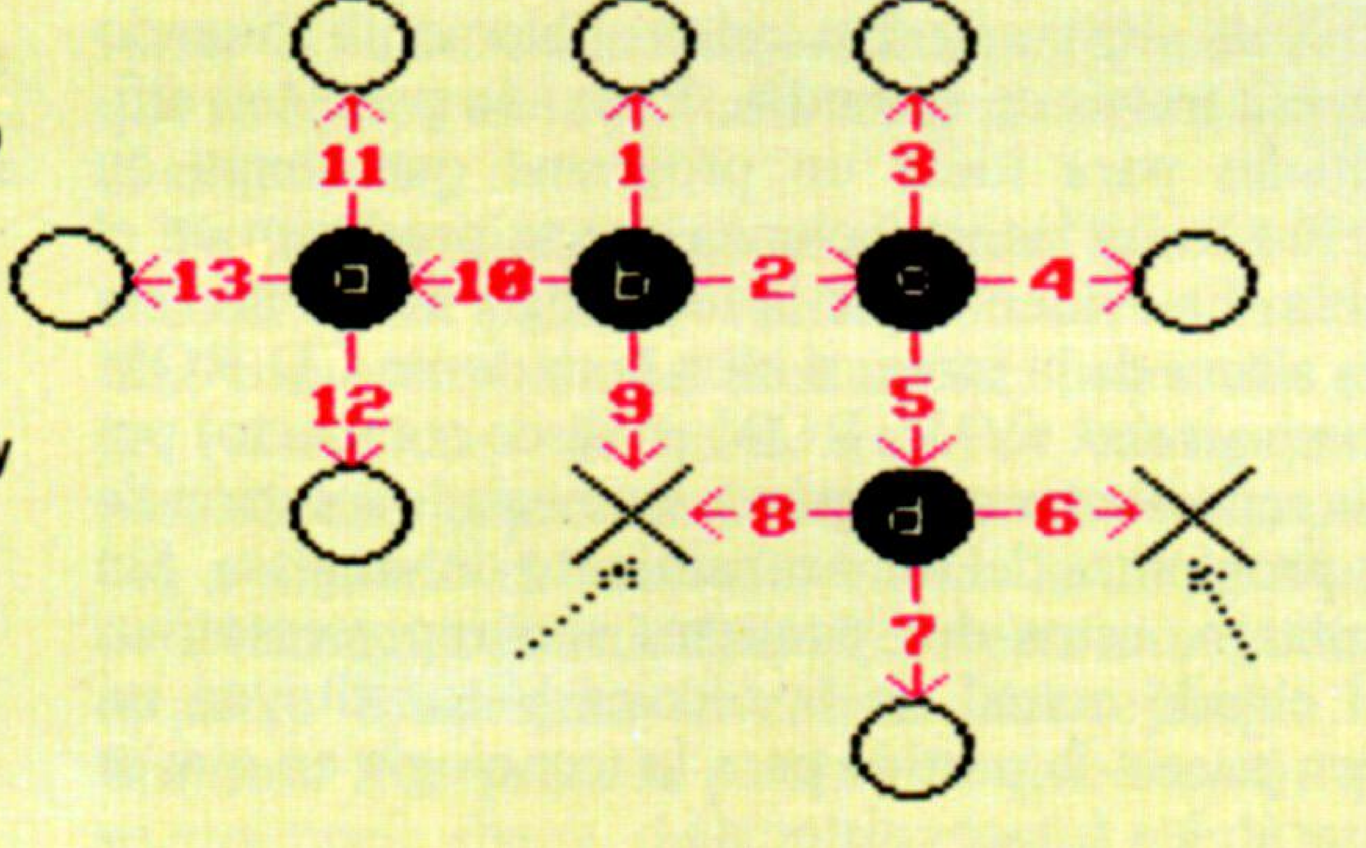
Poder de decisión

Ahora diseñaremos algunos procedimientos que permitan al ordenador «decidir» sus propios movimientos

En este capítulo damos comienzo a la verdadera parte de inteligencia artificial del programa, donde le «enseñamos» al ordenador cómo elegir movimientos razonables. La primera rutina «movimiento del ordenador» le permitirá a éste decidir si algún grupo determinado del tablero se halla o no en problemas. Una vez determinado esto, podemos mostrarle a la máquina cómo seleccionar un movimiento que tenga posibilidades ya sea de poner el grupo a salvo, si fuera uno propio, o bien de atacar y capturar un grupo del adversario.

Antes de comenzar a abordar este problema, veamos en qué forma eligen los movimientos la mayoría de los programas para juegos. En el juego del ajedrez, por ejemplo, se sabe que desde cualquier posición dada del tablero son posibles (como media) aproximadamente 30 movimientos. Tomando esto como base, si un ordenador hubiera de probar 30 posibles movimientos, luego habría de examinar 30 posibles posiciones, decidiendo lo mejor de cada una de ellas. Al intentar decidir la conveniencia de cada movimiento, usted, por supuesto, tendrá que comprobar qué movimientos puede efectuar su oponente en respuesta al suyo: es evidente que un movimiento no es bueno si el oponente puede colocarlo a usted en jaque inmediatamente. Si suponemos que el ordenador va a examinar todos los movimientos que puede efectuar el oponente desde las 30 posiciones que puede alcanzar inicialmente la máquina, luego es necesario examinar algo así como 900 (30 × 30) posibles posiciones. Llevando esto aún otro poco más hacia adelante, podríamos analizar todas las posibles réplicas del ordenador, dando aproximadamente 27 000 posiciones a evaluar, seguidas por 810 000 posiciones al nivel siguiente, y así sucesivamente. Esta evaluación «anticipada» se puede ilustrar gráficamente en forma de *árbol de juego* (similar a un árbol genealó-

Situación crítica
La rutina de evaluación de grupos del programa considera que un grupo se halla en peligro si posee sólo una o dos licencias. El diagrama muestra el orden por el cual se evalúa el grupo de negras, comenzando por la ficha b. Se utilizan dos elementos de matriz, cloc%(1) y cloc%(2), para almacenar las posiciones de las licencias descubiertas, puesto que si el grupo se halla en peligro, el ordenador probablemente querrá jugar en uno de estos puntos



Módulo 4

BBC Micro:

```
  80 movimiento%=movimiento%+1
     :PROCmovimiento_negras
 240 DIM captura%(2),cloc%(2),tloc%(2)
1360 atari1$="          ":atari2$="          ":T$"***"
2530 :
2540 DEF PROCmovimiento_negras
2550 atari1$="          "
2560 posicion%=0
2570 PROCevaluacion_grupos:T$="GP "
2630 IF posicion%=0 THEN ENDPROC
2640 PROCefectuar_movimiento(posicion%,negras%)
2650 PROCmensaje(23,6,"")
2660 ENDPROC
2670 :
2680 REM ****************************************
2690 :
2700 DEF PROCevaluacion_grupos
2710 LOCAL C%,L%,P%,Q%,S%,hi,marcador
2720 FOR P%=17 TO 255
2730    C%=tablero%?P% AND color%
2740    IF C%=0   THEN 2850
2750    PROCcontar(P%,C%) : IF clib%>2 THEN 2850
2760    tloc%(1)=cloc%(1)
2770    tloc%(2)=cloc%(2)
2780    L%=clib% : S%=cstn%
2790    FOR Q%=1 TO L%
2800       IF FNlegalidad(tloc%(Q%),negras%)<>0 THEN
           2840
2810       IF L%=2 AND clib%<3 THEN 2840
2820       marcador=(8*S%/L%-clib%+2*L%)
2830       IF marcador > hi THEN
           hi=marcador:posicion%=tloc%(Q%)
2840       NEXT
2850    NEXT
2860    ENDPROC
2870 :
2880 REM ****************************************
4060 cloc%(1)=0 : cloc%(2)=0
4280 IF clib%<3 THEN cloc%(clib%)=P%
```

Amstrad CPC 464/664:

```
  80 mov%=mov%+1:GOSUB 2540
 240 DIM captura%(2),cloc%(2),tloc%(2)
1360 atari1$="          ":atari2$="          ":T$="***"
2530 :
2540 REM **** rutina movimiento negras ****
2550 atari1$="          "
2560 posicion%=0
2570 GOSUB 2700:T$="GR ":REM evaluar grupos
2630 IF posicion%=0 THEN RETURN
2640 mmp%=posicion%:mmc%=negras%:GOSUB 3630:REM
     efectuar movimiento
2650 mp%=25:mm%=6:o$="":GOSUB 2160:REM
     mensaje
2660 RETURN
2670 :
2680 REM ****************************************
2690 :
2700 REM rutina evaluacion grupos
2710 hi%=-9999
2720 FOR p%=17 TO 255
2730 c%=PEEK(tablero+p%) AND color%
2740 IF c%=0 THEN 2850
2750 cp%=p%:cc%=c%:GOSUB 4040: IF clib%>2 THEN
     2850
2760 tloc%(1)=cloc%(1)
2770 tloc%(2)=cloc%(2)
2780 gl%=clib%:gs%=cstn%
2790 FOR q%=1 TO gl%
2800 lp%=tloc%(q%):lc%=negras%:GOSUB 3890:IF ll%<>0
     THEN 2840
2810 IF gl%=2 AND clib%>3 THEN 2840
2820 marcador=(8*gs%/gl%-clib%+2* gl%)
2830 IF marcador > hi THEN
     hi=marcador:posicion%=tloc%(q%)
2840 next q%
2850 NEXT p%
2860 RETURN
2870 :
2880 REM ****************************************
4060 cloc%(1)=0:cloc%(2)=0
4280 IF clib%< 3 THEN cloc%(clib)=sp%
```

Commodore 64:

```
  80 MOVIMIENTO%=MOVIMIENTO%+1:GOSUB 2540
 240 DIM CAPTURA%(2),CLOC%(2),TLOC%(2)
1360 :A1$="          ":A2$="          ":T$="***"
2530 :
2540 REM RUTINA MOVIMIENTO NEGRAS
2550 A1$="          "
2560 POSIC%=0
2570 GOSUB 2700:T$="GP "
2630 IF POSIC%=0 THEN RETURN
2640 MP%=POSIC%:MC%=NEGRAS%:GOSUB 3630
2650 RETURN
2670 :
2680 REM  **************************************
2690 :
2700 REM RUTINA EVALUACION GRUPOS
2710 HI=-9999
2720 FOR P=17 TO 255
2730 C%=PEEK(TABLERO+P) AND COLOR%
2740 IF C%=0 GOTO 2850
2750 CP%=P:CC%=C%:GOSUB 4040:IF CLIB%> 2 GOTO
     2850
2760 TLOC%(1)=CLOC%(1)
2770 TLOC%(2)=CLOC%(2)
2780 BL=CLIB%:BS=CSTN%
2790 FOR Q=1 TO BL
2800 LP%=TLOC%(Q):LC%=NEGRAS%:GOSUB 3890: IF
     LL%<> 0 GOTO 2840
2810 IF BL=2 AND CLIB%<3 GOTO 2840
2820 SCR=(8*BS/BL-CLIB%+2*BL)
2830 IF SCR<>HI THEN HI=SCR:POSIC%=
     TLOC%(Q)
2840 NEXT
2850 NEXT
2860 RETURN
2870 :
2880 REM  **************************************
4060 CLOC%(1)=0:CLOC%(2)=0
4280 IF CLIB%<3 THEN CLOC%(CLIB%)=SP%
```

Sinclair Spectrum:

```
  80 LET movimiento=movimiento+1: GO SUB
     2540
 240 DIM c(2): DIM i(2): DIM j(2)
1360 LET x$="          ": LET y$="          ":
     LET t$="***"
2530 :
2540 REM rutina movimiento negras
2550 LET x$="          "
2560 LET posicion=0
2570 GO SUB 2700: LET t$="GP "
2630 IF posicion=0 THEN RETURN
2640 LET mmp=posicion: LET mmc=negras: GO SUB
     3630
2650 LEt mp=21: LET mm=7: LET o$="": GO SUB
     2160
2660 RETURN
2670 :
2680 REM  **************************************
2690 :
2700 REM rutina evaluacion grupos
2710 LET hi=-9999
2720 FOR p=17 TO 255
2730 LET c=PEEK(tablero+p)
2735 IF c>3 THEN LET c=c-4: GO TO 2735
2740 IF c=0 THEN GO TO 2850
2750 LET cp=p: LET cc=c: GO
     SUB 4040: IF clib>2 THEN GO TO 2850
2760 LET j(1)=i(1)
2770 LET j(2)=i(2)
2780 LET gl=clib: LET gs=cstn
2790 FOR q=1 TO gl
2800 LET lp=j(q): LET lc=negras: GO
     SUB 3890: IF ll<>0 THEN GO TO
     2840
2810 IF gl=2 AND clib<3 THEN GO TO 2840
2820 LET marcador=(8*gs/gl-clib+2*gl)
2830 IF marcador>hi THEN LET hi=marcador: LET
     posicion=j(q)
2840 NEXT p
2850 NEXT q
2860 RETURN
2870 :
2880 REM  **************************************
4060 LET i(1)=0:LET i(2)=0
4280 IF clib<3 THEN LET i(clib)=sp
```

gico), en el cual las bifurcaciones desde cada nudo muestran los movimientos posibles desde esa posición. Este concepto lo hemos analizado profundamente en nuestra serie sobre inteligencia artificial. Estos números, aun pareciendo enormes, se pueden reducir en cierta forma aplicando diversas técnicas (en especial el *algoritmo alfa-beta*) y resultan manejables para un ordenador de gran velocidad.

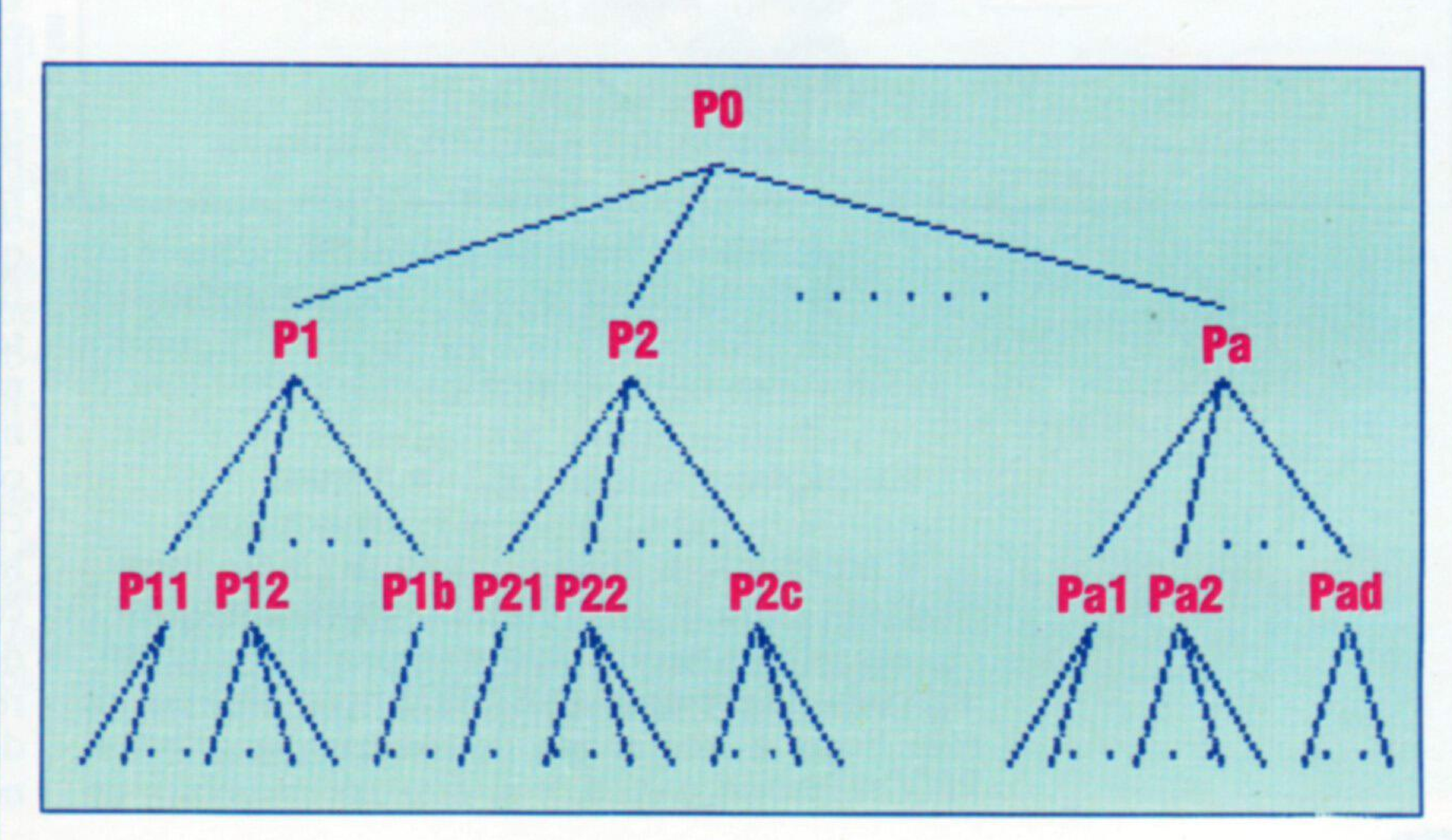
El número promedio de movimientos disponibles desde una posición dada en el juego del *go* (en un tablero de 19 por 19) se calcula en alrededor de 200. En nuestro tablero de 15 por 15, es probable que esta cifra se reduzca a alrededor de 125. Si aplicamos nuestra técnica de anticipación estándar, el crecimiento exponencial da 15 625 posibilidades para las réplicas del oponente, 1 953 125 para el segundo movimiento del ordenador, 244 140 625 posiciones a partir de allí, etc. Con o sin ordenadores veloces, realmente usted no puede esperar que ellos busquen esta cantidad de posibilidades en un tiempo razonable. Además, no es habitual que los maestros del juego examinen posibilidades con hasta 20 o 30 movimientos de anticipación con el fin de decidir el resultado final de una partida determinada.

Una estrategia alternativa

Obviamente, sería absurdo programar nuestro juego del *go* para determinar un movimiento utilizando técnicas de anticipación por «fuerza bruta». Éste es uno de los principales motivos por los que este juego oriental es tan difícil de programar. Una estrategia alternativa consiste en desarrollar una serie de rutinas que examinen la posición actual del tablero, con la esperanza de poder elegir algunos movimientos probables. Finalmente habrá seis rutinas de evaluación, y la evaluación terminará tan pronto cualquiera de las rutinas encuentre un movimiento factible. Por lo tanto, se ha de llamar a las rutinas por el orden correcto. Una variable, posición% (inicialmente cero), se establece en la posición adecuada del tablero si una rutina encuentra un movimiento. Nuestra rutina principal de llamada movimiento_negras será así:

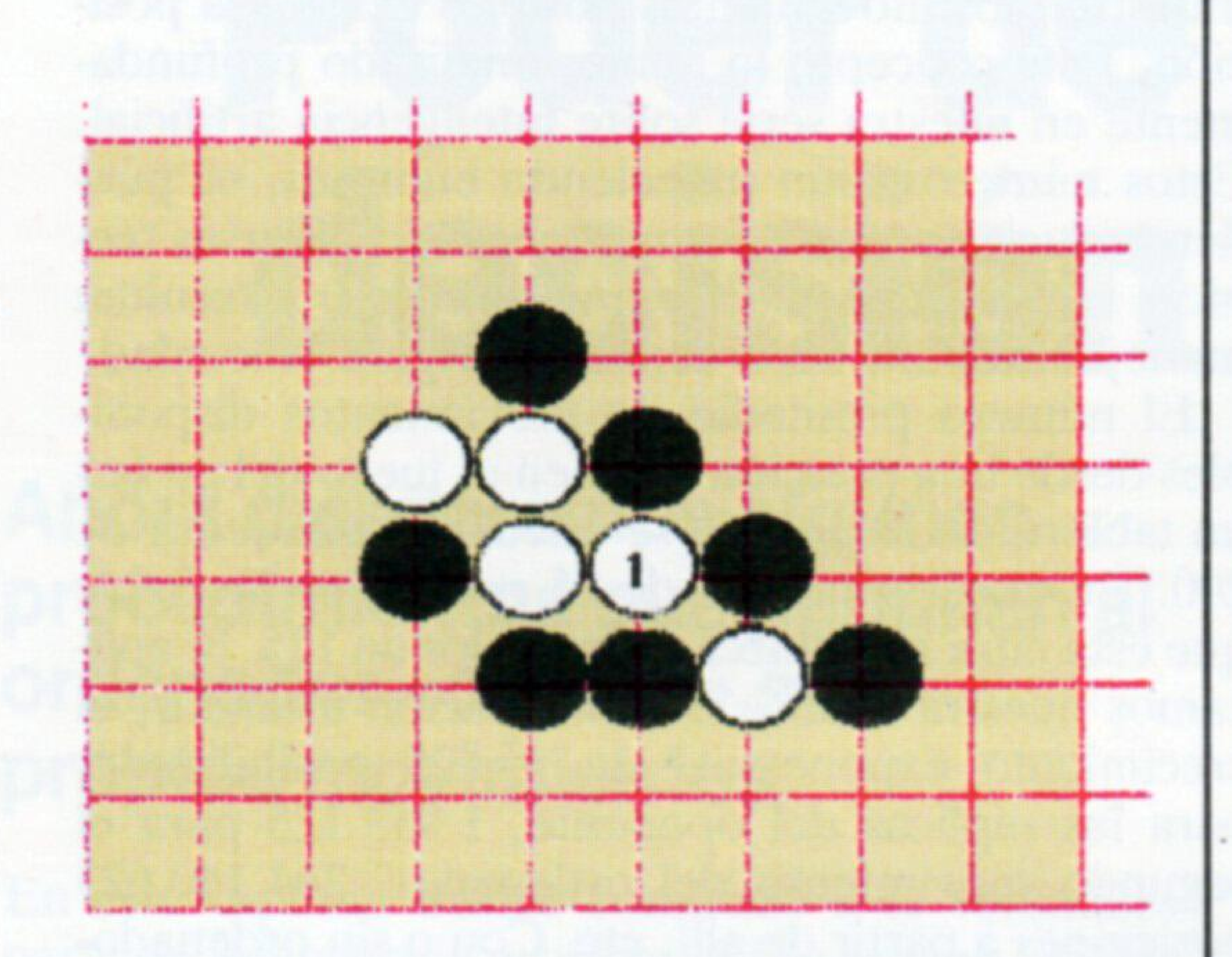
```
LET posicion%=0
CALL                               la primera rutina
                                     de evaluacion
IF posicion%=0 THEN CALL           la segunda rutina
                                     de evaluacion
```

El crecimiento del «go»
Muchos juegos de destreza, como el ajedrez, utilizan árboles de juego de anticipación para decidir el movimiento a realizar. La naturaleza del *go*, sin embargo, significa que se han de considerar alrededor de 200 nuevas ramificaciones tan sólo para analizar un movimiento anticipado; el análisis de dos movimientos implica la comprobación de 40 000 movimientos, y así sucesivamente. Si comparamos esta velocidad de crecimiento con la del ajedrez, podemos ver por qué producir un programa para ordenador para jugar al *go* a un nivel elevado es tan difícil

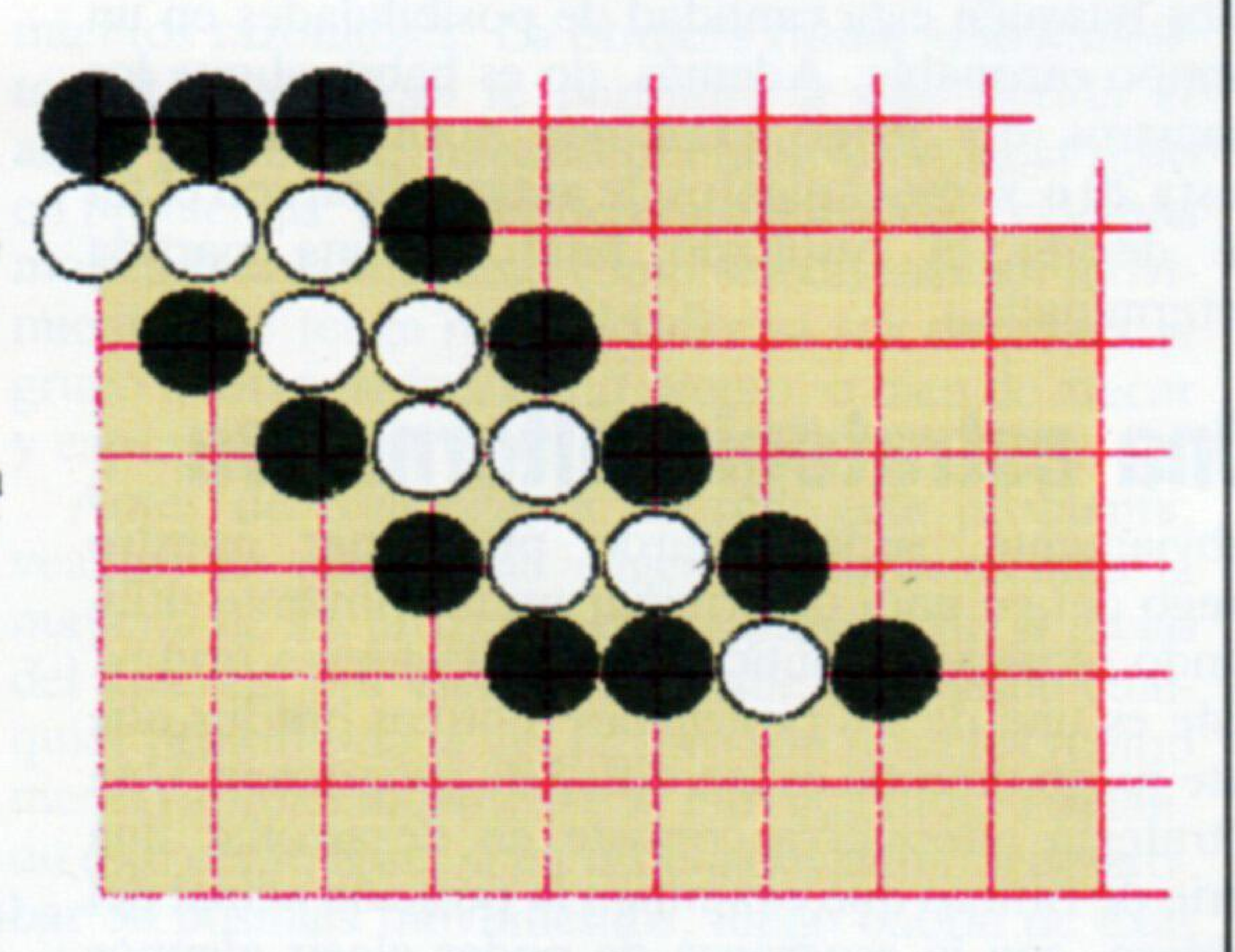


SHICHO

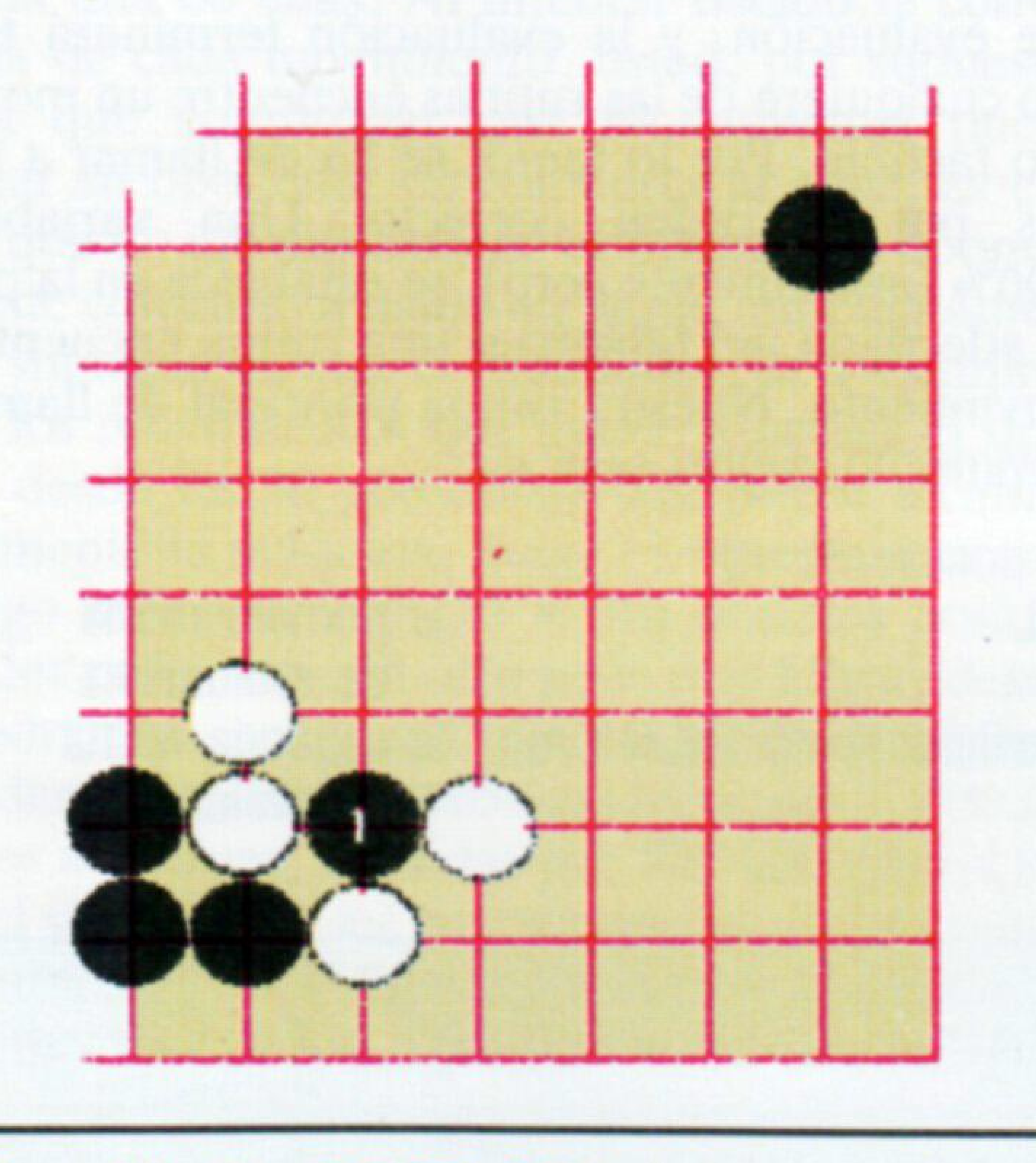
Diagonales peligrosas
La ficha blanca 1 está atacada por las negras y, en vez de sacrificar esta ficha, las blancas intentan extenderse en diagonal hacia arriba y a la izquierda, en un intento por evitar la captura



Fuera de sitio
Las blancas han cometido un grave error al tratar de huir. Finalmente el grupo de blancas en extensión alcanzará el borde del tablero, donde se queda sin sitio y puede ser capturado por las negras. En vez de huir, las blancas deberían haberse preparado para rodear a la ficha original cuando fue atacada por primera vez. Los patrones zigzag en diagonal que se forman durante este tipo de juego se denominan *shicho*



Para romper el «shicho»
En esta situación, las blancas sentirán la tentación de tratar de capturar a la ficha negra marcada como 1, suponiendo que si las negras intentan huir se formará un *shicho* y las negras se quedarán sin sitio en el margen derecho del tablero. Sin embargo, como las negras huyen en diagonal hacia arriba a la derecha, finalmente se unirán a la ficha negra que está sola, rompiendo el *shicho* y permitiendo que las negras escapen a la captura



Caroline Clayton

```
IF posicion%=0 THEN CALL  la tercera rutina
                             de evaluación
        .        .        .
        .        .        .
        .        .        .
IF posicion%=0 THEN CALL  la n rutina
                             de evaluacion
IF posicion%=0 THEN       no se pueden hallar
                             movs. a realizar
```

Tal como se presenta aquí, PROCmovimiento_negras sólo llama a una rutina de evaluación, a saber, PROCevaluacion_grupos, y si ésta no encuentra un movimiento (si posicion% sigue siendo cero), entonces la rutina simplemente termina. En consecuencia, usted aún no puede esperar que el programa juegue una partida en toda la regla, aunque el ordenador intentará defender un grupo si usted lo ataca.

La rutina «evaluación de grupos» siempre tendrá la máxima prioridad en PROCmovimiento_negras, de modo que siempre se comprobará primero. Tiene por finalidad examinar todos los grupos del tablero y contar sus licencias. Si hay algún grupo que sólo tenga una o dos licencias, se considerará que el mismo está en una situación crítica (es decir, a un solo movimiento de «Atari»), de modo que el ordenador intentará ya sea defender un grupo del ordenador, o bien atacar a un grupo enemigo. La primera acción de la rutina es contar las licencias de cada grupo y guardar las posiciones de éstas si el grupo sólo tuviera una o dos. Nosotros queremos guardar estas posiciones porque nuestro movimiento se producirá en alguna de estas posiciones si deseamos ampliar el grupo para salvarlo, o, atacando, rodear el grupo para capturarlo.

En un capítulo anterior señalamos que la rutina PROCcontar cuenta no sólo las fichas de cualquier grupo, sino también las licencias. Añadiéndole la línea 4060 a PROCcontar y la línea 4280 a PROCbuscar, la cuenta de licencias se almacenará en la matriz cloc%(2).

Evaluación de grupos

Ahora se puede definir PROCevaluacion_grupos. El bucle P% comprueba todas las posiciones del tablero. De hallar un grupo, se cuentan sus licencias y, si posee menos de tres, entonces el grupo es crítico, de modo que se entra en el bucle Q%. Éste comprueba la legalidad de colocar una ficha en la(s) licencia(s) y, de ser ilegal, se le asigna un marcador al movimiento. Esta función del marcador (en la línea 2820) se descubrió básicamente mediante ensayo y error y no es necesariamente la mejor; tal vez a usted le interesa experimentar. Observe que como producto secundario de la llamada a FNlegalidad, ahora clib% y cstn% contienen el número de licencias y fichas del grupo, en el supuesto de que se efectúe el movimiento. Por consiguiente, los valores originales de clib% y cstn% se han guardado en L% y S%.

Por razones de simplicidad, PROCevaluacion_grupos considera todas las fichas del tablero, lo que implica que si un grupo contiene más de una ficha, entonces lo contará más de una vez. Este método, ligeramente ineficiente, se podría mejorar asegurando que los marcadores (establecidos durante PROCbuscar para indicar que se ha contado una ficha) no se borren al final de PROCcontar. Observe que los marcadores de licencias se deben borrar al final de cada búsqueda, puesto que grupos diferentes pueden compartir las mismas licencias. Asimismo, usted debe asegurarse de que sólo se dejen los marcadores cuando se esté llamando a la rutina contar desde PROCevaluacion_grupos. Más adelante, cuando se hayan despejado los marcadores, se habrá de guardar la posición de todas las licencias y cuentas correspondientes, y comprobar la legalidad de cada movimiento. En el próximo capítulo añadiremos una rutina de evaluación de «captura» a fin de proporcionar un movimiento razonable cuando no sea preciso atacar o defenderse.



Pila eficaz

La utilización de la «notación polaca inversa» simplifica las operaciones aritméticas y la manipulación de datos

Es de sentido común que no se puede obtener la suma de dos números hasta saber cuáles son. Esto sugiere que, aunque escribamos 2+3 con el + en el medio, un ordenador preferiría disponer de los dos números antes de empezar a preocuparse por el +. En realidad piensa en términos de 2 3 + (o sea, tomar los dos números y después sumarlos).

Nosotros estamos habituados a escribir los operadores aritméticos tales como +, −, * y / en el medio («notación de infijos»), como en 2+2. A menudo refleja el lenguaje («dos más dos son cuatro») y separa claramente los operandos entre sí. Éstas son buenas razones para usar la notación de infijos en un lenguaje para ordenador.

Sin embargo, la notación de infijos también posee sus inconvenientes, que aparecen cuando se desea que el lenguaje sea ampliable. En primer lugar, un operador escrito de esta forma debe tener exactamente dos operandos (argumentos o parámetros), uno a cada lado. Para cualquier otra cosa más, se necesita una notación funcional, como FNf(a, b, c, d, e). En segundo lugar, la notación es intrínsecamente ambigua cuando uno escribe algo como 2+3*5. ¿Qué se hace primero, el + o el *? Esto sólo se puede determinar a partir de un conjunto de reglas adicionales, como la que dice que * tiene «mayor prioridad» que +. Si usted quiere ampliar la notación de infijos a nuevos operadores, también ha de poder ampliar las reglas.

El FORTH se inclina por la solución más simple posible: la que, de todos modos, quería el ordenador. Para *todos* los operadores o funciones (todas las palabras, de hecho), ya sean tradicionales, como +, o nuevas, definidas por el usuario, primero se escriben los operandos y después la palabra: como 2 3 +. Usted puede pensar en esto como si fuera una especie de «notación de libro de recetas»: juntar los ingredientes y luego cocinarlos. Su nombre técnico es *notación polaca inversa*. La notación polaca directa coloca el operador *antes* de los operandos y es la que utiliza el LOGO para las nuevas funciones. Señalemos que todos los números que manipula el FORTH son enteros. Ello se debe a razones de eficiencia, pero significa que la división normalmente no da la respuesta fraccionaria exacta.

Supongamos ahora que tiene una expresión de infijos más compleja, como 2*3+8/4. De acuerdo a la prioridad usual, el *último* operador a realizar será el +, de modo que el resultado final será la suma de 2*3 y 8/4. Por lo tanto, una primera etapa para escribir esto en notación polaca inversa es:

(2*3)(8/4)+

Control de la pila

No siempre es posible acomodar la pila de modo tal que los números queden correctamente ordenados para las operaciones, sin tener primero que manipular sus posiciones. El FORTH ofrece numerosas palabras para la manipulación de la pila:

- **DROP** x – –
  (Elimina la parte superior de la pila)
- **DUP** x – – x,x
  (Duplica la parte superior de la pila)
- **SWAP** x,y – – y,x
  (Invierte los dos elementos superiores de la pila)
- **ROT** x,y,z – – y,z,x
  (Hace rotar los tres elementos superiores de la pila, de modo que el tercer número, que estaba abajo, pasa a quedar arriba)
- **OVER** x,y – – x,y,z
  (Copia encima el segundo número, que se hallaba debajo)
- **PICK** xn, . . .,x2, x1, x0, n – – xn, . . ., x2, x1, x0, xn
  (Igual que OVER, pero se debe proporcionar un número extra (n), para decir qué número es el que se debe copiar arriba. P. ej., 1 PICK es igual que OVER, 2 PICK copia encima el tercer número de debajo, y 0 PICK es similar a DUP)
- **ROLL** xn, . . ., x2, x1, x0, n – – . . ., x2, x1, x0, xn
  (Igual que ROT, pero con núm. extra, como en PICK. 2 ROLL es similar a ROT, y 3 ROLL hace rotar 4 núms. 1 ROLL es igual que SWAP)

El figFORTH carece de PICK y ROLL. El FORTH-79 sí las tiene, pero el número extra que usted proporciona es uno más que en FORTH-83. Por lo tanto, en FORTH-79, 3 ROLL es como ROT

Pero esto todavía no es correcto, porque 2*3 y 8/4 aún están en notación de infijos. Reescribámoslos también:

(2 3 *) (8 4 /) +

Quizá piense que necesita los paréntesis para señalar la forma de agrupamiento, pero en realidad la notación polaca inversa *jamás* necesita paréntesis. El agrupamiento siempre es ambiguo, aun sin ellos. De hecho, el FORTH utiliza paréntesis para algo muy distinto (comentarios), de modo que usted está obligado a eliminar los paréntesis. Ello nos deja con la escritura final de 2*3+8/4 en notación polaca inversa:

2 3 * 8 4 / +

Recuerde que en FORTH los espacios son esenciales.

Ahora podemos ver cómo la notación polaca inversa supera los dos problemas de ampliabilidad de la notación de infijos. En primer lugar, no existe ningún motivo por el cual un operador deba abarcar sólo dos operandos. Es tan fácil escribir un operando como tres o cuatro seguidos del operador. El */ del FORTH, por ejemplo, tiene tres operandos: multiplica los dos primeros entre sí, y divide el resultado por el tercero. Esto se ajusta naturalmente al sistema polaco inverso.

El segundo problema que hallamos en la notación de infijos era que, para eliminar la ambigüedad, requería reglas de prioridad y paréntesis. En la notación polaca inversa, este problema no existe. Le dice al ordenador cómo calcular exactamente el resultado de una forma en absoluto ambigua, sin reglas adicionales ni paréntesis.

Veamos ahora cómo resuelve el ordenador una expresión en notación polaca inversa. La forma más simple de ilustrarlo es con algo como 2 3 +. El FORTH encuentra el 2, lo memoriza, luego encuentra el 3, y lo «recuerda». Cuando encuentra el + sabe (o, mejor dicho, ha de suponer) que ya ha memorizado dos números. Los busca, los suma entre sí y «recuerda» el resultado en caso de que a continuación venga otro cálculo, o bien si se ha de utilizar el resultado con algún otro fin, como, por ejemplo, imprimirlo en la pantalla.

En el capítulo anterior presentamos una breve explicación acerca de la forma en que el FORTH recuerda los números, «empujándolos» en una pila sobre una base LIFO (último en entrar, primero en salir). A los programadores de lenguaje máquina este concepto les resultará familiar; pero ofrecemos otro ejemplo para quienes encuentren confuso el ejemplo de la pila. Para ver cómo el FORTH «recuerda» un número, piense en cómo lo haría usted a mano utilizando una pila de naipes sobre la mesa. Para recordar un número, lo escribe en un naipe nuevo y lo coloca encima de la pila. Para +, usted debe sacar los dos naipes de encima y usar los números consignados en ellos, pero uno no altera el resto de la pila. Para 2 −3 * 8 4 / + (para (2 * −3) + (8/4)), la pila evoluciona tal como se indica en el diagrama. Observe cómo, en la etapa 6 del procedimiento, el naipe que lleva escrito −6 queda sin modificar mientras se divide 8 por 4.

Llegados a este punto, hemos de destacar que las variables se tratan de forma bastante diferente a los enteros utilizados en este ejemplo. Una variable deja su dirección en la pila. Ésta se puede luego convertir al valor de la variable (usando @, la ins-

Poniéndolo encima
Este otro ejemplo muestra cómo los números se «empujan» en la pila y se «sacan» de ella, en la ejecución de 2 −3* 8 4 / (2* −3+8/4), en notación de infijos)

Etapa 1
Se coloca el número 2 en la parte superior de la pila...

Etapa 2
...y después se coloca −3 encima del 2

Etapa 3
la palabra * toma dos números de la parte superior de la pila, los multiplica entre sí y coloca el resultado en la pila

Etapa 4
Ahora se pone el 8 en la pila...

Etapa 5
...seguido del 4...

Etapa 6
...y luego / toma dos números de la pila (observe que el −6 no sufre ninguna modificación), se opera sobre ellos y se devuelve el resultado a la parte superior de la pila

Etapa 7
+ saca ahora de la pila los dos números de arriba, los suma y vuelve a colocar el resultado

Caroline Clayton



trucción «traer»), o se puede emplear para actualizar la variable mediante la instrucción ! («almacenar»).

El ejemplo aritmético de la pila demuestra claramente la idoneidad de la notación polaca inversa. El FORTH mantiene su propia imitación interna de la pila de naipes (denominada *pila de datos*) y luego, con cada número o palabra que encuentre, el ordenador puede hacer algo definido sin preocuparse por lo que ocurrió antes o por lo que ocurrirá después.

He aquí otro ejemplo de cómo utilizar la pila. La palabra sustituye a la parte superior de la pila, sea lo que fuere, por sí misma duplicada. Se define:

```
:2* (n -- n*2)
  2 *
;
```

Observe el empleo de los paréntesis para proporcionar comentarios y el símbolo --; ambos se describen en el recuadro «Tomando notas». A modo de ejemplo del empleo de nuestra nueva palabra, 2, la entrada:

```
19 2* .
```

visualiza 38 como resultado. Observe que el símbolo . es la palabra del FORTH para PRINT. Saca la parte superior de la pila y la visualiza en la pantalla. Usted puede ver cómo el * de la definición de 2* espera al menos dos números en la pila. El segundo es el 2 incluido en la definición, pero se espera que el primero (19, en nuestro ejemplo) ya esté en la pila cuando se utilice 2*.

Una de las ventajas de las operaciones basadas en pila es que permiten que una operación produzca más de un resultado. Por ejemplo, la palabra /MOD deja *dos* números en la pila: el «cociente» (respuesta) y el «resto» tras una división. Podríamos expresar esta operación de otra forma, utilizando la notación -- del siguiente modo:

```
m, n -- resto, cociente de m/n
```

Esto sería imposible con la notación de infijos, pero con una pila resulta bastante natural.

La responsabilidad directa de la pila se halla en manos de una potente configuración del FORTH, pero también puede presentar dificultades. Por ejemplo, algunas veces los números tal y como los proporciona la pila no se hallan por el orden correcto y, para reacomodarlos, será preciso efectuar algunas manipulaciones con la pila. En el recuadro «Control de la pila» vemos algunas de las palabras estándares que se utilizan con este fin, y ahora vamos a ver como ejemplo un caso en el cual sería necesario manipular la pila. La palabra */ definida abajo no es tan eficaz como la que se proporciona como estándar (que se cuida de dar la respuesta correcta aun cuando la multiplicación dé un resultado demasiado elevado como para caber en la pila), pero muestra en acción a las palabras manipuladoras ROT y SWAP:

```
:*/ (x,y,z -- x*y/z)
   ROT ROT*
   SWAP /
;
```

He aquí cómo funciona para el ejemplo 4 3 6 */:

Operación de */	Pila (Arriba →)
Al comienzo de */	4, 3, 6
ROT	3, 6, 4
ROT	6, 4, 3
*	6, 12
SWAP	12, 6
/	2
	Vacía (y se visualiza 2)

Afortunadamente, los detalles precisos de la manipulación de la pila por lo general se disimulan debajo de la alfombra ocultándolos dentro de las definiciones de palabras.

Tomando notas

Ha de tener sumo cuidado en colocar en la pila exactamente los números que necesite una palabra. Si coloca pocos, utilizará algunos de más abajo que supuestamente habrían de quedar inalterados, y si coloca muchos, los números extras se situarán en medio. Para decir con claridad lo que una palabra necesita de la pila, y lo deja tras de sí, puede utilizar la notación "--" y después una lista de lo que la palabra deja. P. ej.:

Palabra:	Efecto:
+	m, n -- m+n
.	m --
*/	x,y,z -- x*y/z

Para cada lista, siempre se enuncia en último lugar la parte superior de la pila. La notación "--" no forma parte del lenguaje FORTH, sino que es una configuración añadida que nos permite comprender mejor los listados. Le resultará muy útil incluirla en las definiciones de palabras del FORTH, en los comentarios (encerrados entre paréntesis). Recuerde que es necesario incluir un espacio tras el primer paréntesis, puesto que es una palabra que significa: «ignorar todo cuanto haya hasta el paréntesis de cierre».

Visualizando la pila

Una palabra que le resultará muy útil es .S, que visualiza la pila. No está incluida en el FORTH estándar, pero en FORTH-83 puede definirla así:

```
:   .S (--)
    ( visualiza la pila, de abajo arriba)
    0 DEPTH 1 - DO
       I PICK.
    -1+LOOP
;
```

En FORTH-79, debido a que PICK trabaja de modo diferente, se debe reemplazar la tercera línea por la siguiente:

```
1 DEPTH DO
```

DEPTH (-- profundidad pila) le dice cuántos números había en la pila antes de que efectuara DEPTH. (El figFORTH no dispone de DEPTH ni de PICK, de modo que en ese dialecto esta definición no funciona)

¿Un poco de ROM?

La Interface 1 puede serle útil para añadir nuevas instrucciones al BASIC del Spectrum. Veamos la «teoría»

Para añadir nuevas instrucciones del BASIC hay que «engañar» a las rutinas de tratamiento de error del Spectrum para que «traguen» algo que normalmente no lo harían. Es, pues, un buen comienzo de nuestro análisis el estudio de cómo se detectan los errores en el Spectrum.

Cuando se digita una línea del BASIC y se pulsa ENTER, la línea es inmediatamente investigada por el intérprete del BASIC por si no fuera sintácticamente correcta. En esta fase se detectan errores tales como la omisión de los nombres de las variables después de instrucciones NEXT, empleo incorrecto de parámetros, etc. Ésta es la causa que impide entrar en este ordenador líneas «torcidas» del BASIC. Si la investigación es positiva se coloca en el lugar adecuado dentro del programa caso de ser una línea de programa, o se ejecuta si es una instrucción en modo indirecto.

Si la línea no es correcta, se ejecutará una instrucción RST#8, seguida de una DEFB, que indica al OS la naturaleza del error cometido. Esto provoca la visualización de un ? parpadeante, del que indudablemente ya habrá tenido usted noticia.

Tras la ejecución de una sentencia, sea en modo directo, sea como línea de programa, se vuelve a comprobar la sintaxis de la línea. Así, en el curso de esta segunda investigación se evalúan las expresiones numéricas contenidas en la línea y se ejecuta mediante llamadas a las correspondientes rutinas de la ROM. En este punto son detectados errores del tipo No Such Variable (No hay tal variable).

El primer chequeo, conocido como *chequeo sintáctico,* no ejecuta la instrucción, sino que sólo asegura su validez sintáctica. El segundo, o *chequeo en fase de ejecución (run-time check),* ejecuta la sentencia. No obstante, cuando se acopla al sistema el hardware de la Interface 1 (IF1), este proceso se altera. Como ya hemos tenido ocasión de ver, el hardware de la Interface 1 se encarga de que siempre que se dé un acceso a la dirección #08 o #1708, la ROM sombra esté paginada y se desconecte la ROM principal. Se dice que ahora el sistema del Spectrum está operando en el «ámbito de la ROM sombra» y la situación se prolonga hasta que hay un acceso a la dirección #700 de la ROM de la IF1 (en este momento se desactiva la ROM sombra y vuelve a reactivarse la ROM principal).

De esta descripción aparece claro que en cuanto se dé un error, la ROM sombra queda paginada y tiene lugar una segunda investigación sobre la condición causante del error. Si el error fue provocado por una instrucción como CAT o FORMAT, que son legales cuando se ha conectado la IF1, éste es tratado por la ROM de la IF1 y se produce un retorno a la ROM principal.

Si la condición del error no fue motivada por ninguna de estas instrucciones, se produce un salto a la dirección contenida en la variable de sistema de la ROM IF1 llamada VECTOR (que se encuentra en las direcciones 23735 y 23736). VECTOR contiene la dirección del tratador de errores de la ROM sombra. Es importante observar que tal dirección puede diferir según las versiones de la ROM IF1. Nosotros hemos empleado la primera versión de la ROM IF1. En el próximo capítulo examinaremos algunas de las principales diferencias entre las distintas versiones de la ROM sombra y le facilitaremos un método para identificar la versión que usted está usando.

Con la primera versión de la ROM IF1, la variable VECTOR contiene el valor #1F0, por lo que una situación de error que reporte una llamada a esta rutina conducirá al editor en tiempo de «chequeo sintáctico» para que sea editado el paso irregular de la línea BASIC que se investiga. En consecuencia, alterando la dirección contenida en VECTOR, desviaremos al tratador de errores de tal modo que ejecute una rutina hecha por nosotros en lugar de la rutina de error habitual. En el próximo capítulo veremos cómo se hace. Pero antes examinemos algunas rutinas de la ROM sombra que nos serán útiles en este proceso.

#0010

Esta rutina permite llamar a la rutina principal cuando se halla en el entorno de la ROM sombra. Es el único medio de llamar a la rutina principal, ya que habremos de poner sumo cuidado en la paginación y despaginación de la ROM principal. Se emplea de la siguiente manera:

```
RST   #0010
DEFW  direccion ;dir. de la rutina por llamar
```

#0020

Esta rutina es la versión para ROM sombra de RST#8 de la ROM principal. Funcionará escribiendo RST #0020, seguido de un solo byte que indique el mensaje de error que ha de ser generado. Los varios mensajes de error se muestran en la siguiente tabla:

#0028

Permite generar un mensaje de error «normal» (es decir, generado en la ROM principal y no en la





De entre las sombras

Además de las facilidades del microdrive, el RS232 y conexión a red que ofrece la Interface 1, la ROM sombra tiene también un buen puñado de útiles rutinas, entre las que se cuenta el convertidor decimal a hexadecimal (que parece no fue asumido por ninguna de las restantes subrutinas del sistema). Para sacar el mayor rendimiento de la interface, recomendamos el libro *Spectrum shadow ROM disassembly*, que proporciona minuciosos detalles de las rutinas de la ROM sombra, así como unos eficaces listados de utilidades



Byte	Mensaje de error
00	Sin sentido en BASIC
01	Núm. no válido de corriente
02	Expresión no válida de dispositivo
03	Nombre no válido
04	Número no válido de drive
05	Número no válido de estación
06	Falta nombre
07	Falta número estación
08	Falta número drive
09	Falta velocidad baudios
10	Error confusión encabezam.
11	Corriente ya abierta
12	Escritura a un archivo lectura
13	Lectura a un archivo escritura
14	Drive protegido contra escritura
15	Microdrive completo
16	No hay microdrive
17	Archivo no hallado
18	Error código enganche
19	Error en CODE
20	Error en MERGE
21	Verificación no lograda
22	Tipo erróneo de archivo
255	Programa concluido

ROM sombra). Antes de llamar a esta rutina, se carga la dirección 23610 con el código de error correspondiente.

#01F0

Genera el cursor parpadeante de *syntax error* mientras se realiza el chequeo sintáctico.

#05B7

Es la rutina a la que se llama para indicar que la sintaxis de la instrucción ha sido comprobada. También comprueba el final de la sentencia BASIC. Más tarde estudiaremos su empleo detalladamente.

#05C1

Empleada para finalizar la ejecución en la ROM sombra, devuelve el control a la ROM principal.

#0700

Provoca un retorno al ámbito de la ROM principal. Puede ser considerada como un medio de despaginar la ROM sombra y de paginar la ROM principal. Examinemos, por último, el código de enganche correspondiente:

- *Código de enganche 50:* Como todos los códigos de enganche *no* ha de usarse en el entorno de la ROM sombra. Le permite a usted llamar a las rutinas de la ROM sombra desde el interior de la ROM principal, y sólo ha de usarse si usted posee un buen conocimiento de las direcciones dentro de la ROM sombra. Se entra con la dirección de la rutina de ROM sombra contenida en la rutina del registro HL, como muestra el siguiente código:

```
LD      HL, direccion;direccion de rutina
LD      (23789), HL
RST     #8
DEFB    50
```

Hay más temas de interés sobre el entorno de la ROM sombra. De lo dicho debe quedar claro que todas las instrucciones *restart* del Z80 son diferentes. El teclado no es explorado cuando nos hallamos en la ROM sombra, ni la variable de sistema FRAMES es incrementada. Consecuentemente, todo espacio de tiempo transcurrido en la ROM sombra tiende a hacer más lenta la variable FRAMES y a hacerla poco fiable en casos de temporización.

A veces un problema que puede presentarse está en el uso de la *calculadora de punto flotante* (*floating point calculator:* FPC) desde la ROM sombra. La rutina que se encuentra en #0010 en la ROM sombra parece incapaz de asumir la llamada a FPC, lo que quiere decir que es mejor salir de la ROM sombra y trabajar en la ROM principal para usarla. Si después usted desea volver a entrar la ROM sombra, lo puede hacer empleando el código de enganche 50.

Empleando un PEEK

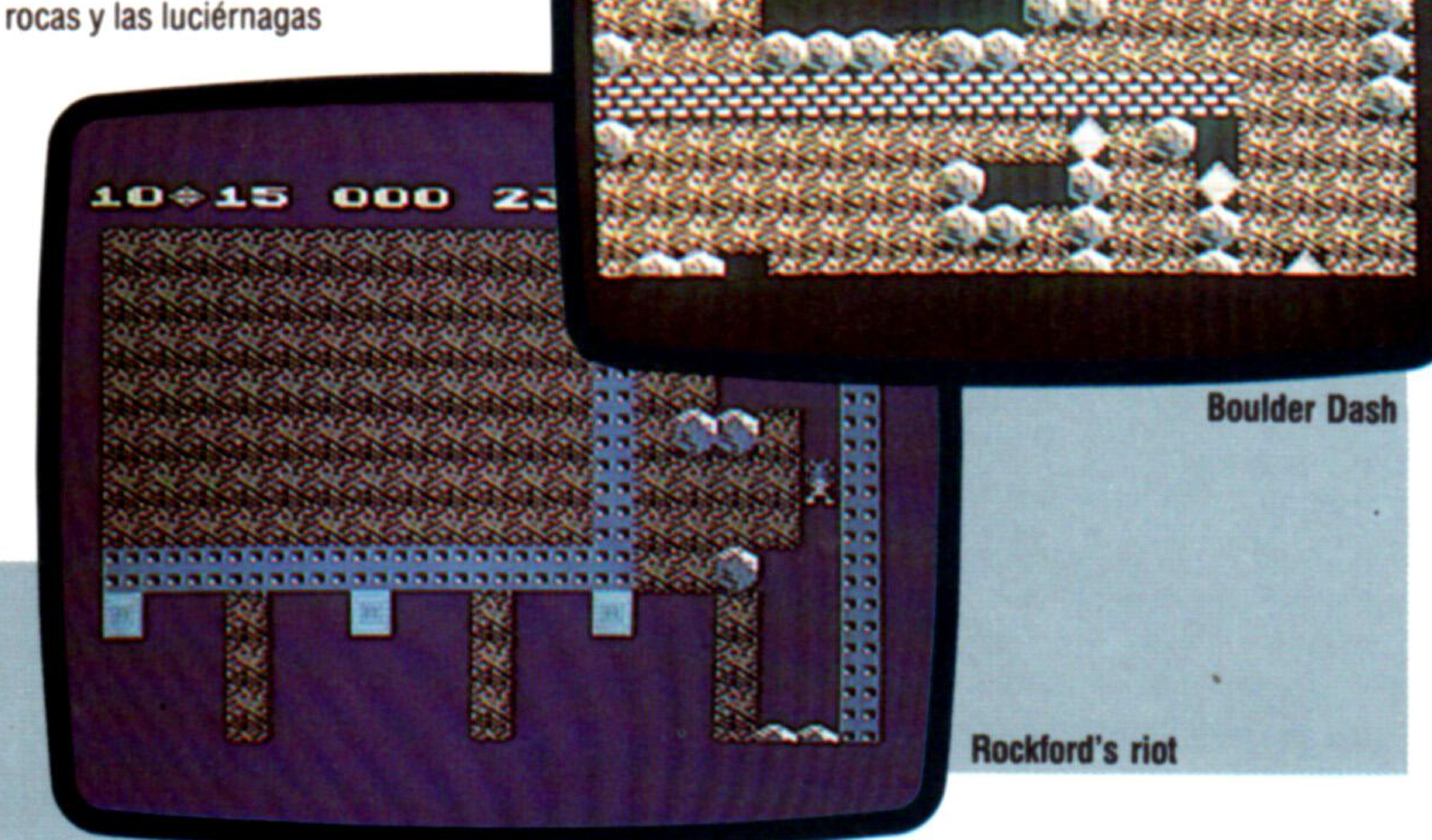
Puede que a usted le agrade examinar algunas de las rutinas aquí mencionadas. El desensamblado de programas ajenos puede ser una experiencia interesante e instructiva y ayudarle a emplear con mucho mayor eficacia las rutinas que se le ofrecen. Por desgracia, directamente desde el BASIC no es posible leer (PEEK) la ROM sombra, ya que sólo está paginada cuando actúa la Interface 1. Este breve programa en BASIC cargará cualquier fragmento de la ROM sombra en la RAM, donde usted sí que podrá examinarlo como guste.

```
10   GO SUB 200
20   INPUT "Cuantos bytes desea ampliar
     (CLEAR)?";b
30   CLEAR r-(b+24)
40   GO SUB 200
50   FOR n=1 TO 23
60   READ d: POKE r+n,d
70   NEXT n
80   INPUT "Direccion inicio en ROM Sombra?";s
90   INPUT "Numero de bytes por copiar?";b
100  LET z=s: GO SUB 300
110  POKE r+12,l:POKE r+13,h
120  LET z=b: GO SUB 300
130  POKE r+18,l:POKE r+19,h
140  LET z=r+9: GO SUB 300
150  POKE r+2,l: POKE r+3,h
160  LET z=r+24: GO SUB 300
170  POKE r+15,l: POKE r+16,h
180  RANDOMIZE USR (r+1)
190  PRINT "Datos almacenados en";r+24
195  STOP
200  LET r=PEEK 23730+256*PEEK 23731:
     RETURN
300  LET h=INT (z/256):LET l=z-256*h:RETURN
400  DATA 33,0,0,34,237,92,207,50,
     225,225,33,0,0,17,0,0,1,0,0,237,176,
     199,201
```

Tesoros ocultos

Comentaremos dos programas de juegos procedentes de Estados Unidos que posiblemente se constituyan en un éxito a nivel mundial

Búsqueda temeraria
Rockford's riot se lanzó a poco de aparecer *Boulder Dash*, que obtuvo un enorme éxito. Al producir la continuación, los programadores se ciñeron a la sencilla fórmula del primer paquete. El objetivo de ambos juegos consiste en excavar la tierra, recogiendo joyas al tiempo que se evitan peligros, como los que representan las rocas y las luciérnagas



Boulder Dash

Rockford's riot

A pesar de que el mercado de software de juegos se ha convertido en una industria multimillonaria, produciéndose literalmente miles de juegos diferentes cada año, son muy pocos los que llegan a provocar impresiones duraderas. Con excesiva frecuencia, aun los juegos de más éxito caen en el olvido a los pocos meses de su lanzamiento. Existen, sin embargo, un puñado de juegos que han tenido una continuación. En Gran Bretaña, por ejemplo, dos de tales juegos son *Manic miner* y su segunda parte, *Jet set Willy*. Este último ha tenido tal éxito que su editora, Software Projects, ha cedido a las presiones del público y ha reeditado el juego con la adición de algunas habitaciones.

En Estados Unidos han surgido ávidos seguidores de Rockford, el héroe de *Boulder Dash* (que también es un juego de «minería») y su éxito ha dado origen a una continuación, denominada *Rockford's riot* (en Gran Bretaña) o *Boulder Dash II* (en Estados Unidos). El nudo de la acción en ambos juegos lo constituye recoger joyas que se hallan enterradas; Rockford debe excavar el terreno para poder alcanzarlas.

Por supuesto, un juego para ordenador estaría incompleto si no hubiera algún obstáculo en el camino. El problema fundamental son las grandes piedras que se hallan diseminadas alrededor de la pantalla y que impiden que Rockford acceda directamente a las joyas. Si cae alguna piedra sobre Rockford, el jugador pierde una vida, aun cuando sea posible que nuestro héroe permanezca de pie con una roca sobre la cabeza. Aunque parecería que las enormes piedras son una molestia, también tienen su utilidad.

Otra de las amenazas son las luciérnagas, criaturas incapaces de horadar la tierra pero que pueden desplazarse por los túneles ya existentes o por los que excava Rockford. Son mortales si se introducen dentro de un cuadrado de Rockford; pero, afortunadamente, tienen su «talón de Aquiles». Primero, cuando disponen de una serie de caminos en los que entrar, siempre optan por la ruta situada más a la izquierda. Segundo, son tan susceptibles de que se les caiga una roca encima como lo es el propio Rockford. Por lo tanto, usted puede predecir hacia dónde se dirigen las luciérnagas, esperar a que lleguen al final de un túnel y, cuando aparezcan, arrojarles una roca encima.

Esta faceta del juego proporciona la solución para otro problema. A menudo las joyas estarán detrás de una pared o de otro obstáculo que le impida el paso a Rockford. Arrojando una roca sobre una luciérnaga que se halle junto a una pared, usted hará explotar tanto a la criatura como a la barrera, donde quedará un agujero.

Para llegar al siguiente nivel se han de recoger una cierta cantidad de joyas, que activarán una puerta situada en algún lugar de la pantalla, a través de la cual pasará Rockford. A los jugadores que no estén familiarizados con el juego esto les resultará desconcertante en un primer momento, porque parece no haber suficientes joyas diseminadas por ahí para que Rockford las recoja. Por ejemplo, ambos juegos contienen una «ameba», una masa verde que se dilata lentamente hasta llenar la pantalla. Sin embargo, si se la rodea con rocas para que no pueda extenderse, la ameba alcanzará una «masa crítica» y se cristalizará en forma de joyas. En otras pantallas, las joyas se crean arrojando piedras sobre objetos determinados.

Gran parte de la acción depende de la rapidez de reflejos, pero la marca de un buen juego es que requiera asimismo movimientos bien pensados.

Los gráficos de *Boulder Dash* y *Rockford's riot*, que son muy similares, son excelentes. Las pantallas se crean en modalidad multicolor y tanto el enrollado de la pantalla como el control del usuario son impecables. Otra característica de estos juegos que hace que jugarlos sea una delicia, es la atención que se ha prestado a los detalles al diseñar las visualizaciones en pantalla.

Boulder Dash y Rockford's riot: Para el Commodore 64 y el Sinclair Spectrum
Editado por: Beyond Software, Competition House, Farndon Road, Market Harborough, LE16 9NR, Gran Bretaña
Autores: Peter Liepa y Chris Gray
Formato: Cassette
Palanca de mando: No se necesita

mi computer

Editorial Delta, S.A.



10098

9 788485 822836